TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG - HCM

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO

MÔN: HỆ ĐIỀU HÀNH

Project 01: Quản lý tập tin

Giảng viên: Lê Viết Long

Sinh viên thực hiện:

1. Lê Kiệt: 19120554
2. Hồ Hữu Ngọc: 19120602
3. Nguyễn Phát Minh: 19120586
4. Hoàng Mạnh Khiêm: 19120543

MỤC LỤC

I. BẢNG PHÂN CÔNG

Họ Và Tên	MSSV	Công Việc	Mức độ hoàn thành
Hồ Hữu Ngọc	19120602	Đọc và in ra thông tin trong boot sector FAT32. Phụ trách chính hàm **readBootSectorFat32**	100%
		Đọc và in ra thông tin trong partition boot sector NTFS. Phụ trách hàm chính hàm **readBootSectorNTFS**	100%
Hoàng Mạnh Khiêm	19120543	Đọc và hiển thị nội dung tập tin (**hàm readData**)	100%
		Phụ trách tổ chức kiểu dữ liệu **struct Fentity**	100%
		Phụ trách chính chỉnh sửa và tối ưu hàm **readXdet**, **convertExtension**, **readFat**	100%
		Viết, chỉnh sửa hàm main và tổ chức luồng chạy chương trình	100%
Lê Kiệt	19120554	Đọc và hiển thị cây thư mục gốc RDET và cây thư mục con SDET (phụ trách chính hàm **readXdet**, **printXdet**, **readFileName**)	100%
		Phụ trách tổ chức kiểu dữ liệu **struct Root** Đọc bảng FAT (**readFat**)	100%
		Viết **menu** và **subMenu** cho chương trình	100%
Nguyễn Phát Minh	19120586	Đọc và in ra thông tin trong boot sector FAT32. Phụ trách chính hàm **readBootSectorFat32**	100%
		Đọc và in ra thông tin trong partition boot sector NTFS. Phụ trách hàm chính hàm **readBootSectorNTFS**	100%

- Đánh giá mức độ hoàn thành project: 80%

II. ĐỌC BOOTSECTOR

1. CÁC HÀM HỖ TRỢ CHO PHẦN NÀY

- Hàm **hextodec**: Hàm này truyền vào một chuỗi string là chuỗi hexa và trả về một kiểu số nguyên cơ số 10

```
66  int hextodec(string s)
67  {
68      char* p;
69      long n = strtoul(s.c_str(), &p, 16);
70      return n;
71  }
```

- Hàm **sectorToDec**: Đọc thông tin dãy byte từ tham số **sector**, bắt đầu đọc từ vị trí **start** và lấy **count_bytes** (byte) muốn đọc, sau đó gọi hàm **hextodec** để trả về số nguyên cần đọc từ dãy byte vừa lấy

```
73  int sectorToDec(BYTE sector[512], int start, int count_bytes)
74  {
75      stringstream hexa;
76      string s;
77      for (int i = 0; i < count_bytes; i++)
78      {
79          hexa << hex << setfill('0') << setw(2) << int(sector[start + count_bytes - 1 - i]);
80      }
81      s = hexa.str();
82      return hextodec(s);
83  }
```

2. ĐỌC BOOTSECTOR CỦA FAT 32

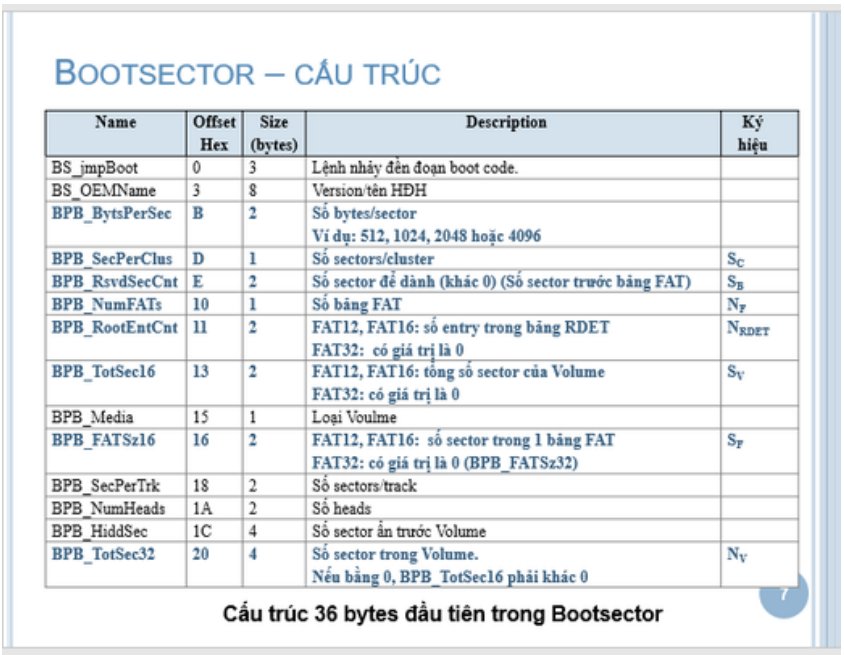
- Hàm **readBootSectorFat32**: Xuất ra những thông tin cần lấy của ổ đĩa Fat 32 dưới dạng số nguyên cần lấy hoặc kiểu ASCII bằng cách gọi hàm sectorToDec và truyền vào các chỉ mục cần thiết.

```
85  void readBootSectorFat32(BYTE sector[512]) {
86      cout << "\tOS version: ";
87      for (int i = 0; i < 8; i++)
88          cout << char(int(sector[3 + i]));
89
90      cout << "\n\tNumber of sectors/fat: " << int(sector[13]);
91      cout << "\n\tNumber of sectors/cluster: " << sectorToDec(sector, 13, 1);
92      cout << "\n\tNumber of bytes/sector: " << sectorToDec(sector, 11, 2);
93      cout << "\n\tNumber of FAT table(s): " << sectorToDec(sector, 16, 1);
94      cout << "\n\tNumber of entries in 1 FAT table: " << sectorToDec(sector, 17, 2);
95      cout << "\n\tVolume type: " << int(sector[21]);
96      cout << "\n\tNumber of sectors/track: " << sectorToDec(sector, 24, 2);
97      cout << "\n\tNumber of heads: " << sectorToDec(sector, 26, 2);
98      cout << "\n\tBPB_HiddSec:  " << sectorToDec(sector, 28, 4);
99      cout << "\n\tBPB_FATSz32: " << sectorToDec(sector, 36, 4);
100     cout << "\n\tExit flag: " << sectorToDec(sector, 40, 2);
101     cout << "\n\tFAT32 verison: " << sectorToDec(sector, 42, 2);
102     cout << "\n\tFirst cluster index in RDET: " << sectorToDec(sector, 44, 4);
103     cout << "\n\tBPB_FSInfor: " << sectorToDec(sector, 48, 2);
104     cout << "\n\tBPB_BkBootSec: " << sectorToDec(sector, 50, 2);
105     cout << "\n\tReserved: " << sectorToDec(sector, 52, 12);
106     cout << "\n\tBPB_DrvNum: " << sectorToDec(sector, 64, 1);
107     cout << "\n\tBPB_Reservedl: " << sectorToDec(sector, 65, 1);
108     cout << "\n\tBPB_BootSig: " << sectorToDec(sector, 66, 1);
109     cout << "\n\tVolume serial number: " << sectorToDec(sector, 67, 4);
110     cout << "\n\tFAT type: ";
111     for (int i = 0; i < 8; i++)
112         cout << char(sector[82 + i]);
113 }
```

- Ở đây để lấy được index cần đọc ta dùng công thức: index = row * 16 + col (row: Vị trí dòng , col: Vị trí cột của offset cần lấy). Ví dụ:

- Để lấy được 2 byte tại offset 0B (tương ứng sẽ là dòng 0 cột 11), thì index tương đương với index = 0*16 + 11 = 11. Như vậy khi gọi hàm **sectorToDec(sector, 11,2)** có thể lấy được 2 byte tại offset B.
- Để lấy được 1 byte tại offset 10 (tương ứng sẽ là dòng 1 cột 0) thì index tương đương với index =1*16 + 0 = 16. Như vậy khi gọi hàm **sectorToDec(sector, 16,1)** để có thể lấy được 1 byte tại offset 10.

- Ngoại trừ kiểu WORD (tên ổ đĩa và loại ổ đĩa) được ép về kiểu char từng kí tự thì những offset khác ta sẽ truyền vào hàm sectorToDec một index thích hợp để có thông tin cần lấy dưới dạng số thập phân.
- Để đọc thông tin về bootsector của Fat 32, đầu tiên ta đọc 512 byte đầu tiên. Vì Fat 32 lưu trữ dữ liệu theo kiểu little-endian nên ta sẽ đọc và in lần lượt các thuộc tính dựa vào bảng dưới đây (trong slide) và dùng hàm **sectorToDec** để lấy được thông tin cần lấy. Dựa theo nguyên tắc vừa trình bày, index được tính tương tự từ **Offset Hex** và **Size (bytes)** trong 2 hình dưới



BOOTSECTOR – CẤU TRÚC

Name	Offset Hex	Size (bytes)	Description	Ký hiệu
BS_jmpBoot	0	3	Lệnh nhảy đến đoạn boot code.	
BS_OEMName	3	8	Version/tên HĐH	
BPB_BytsPerSec	B	2	Số bytes/sector Ví dụ: 512, 1024, 2048 hoặc 4096	
BPB_SecPerClus	D	1	Số sectors/cluster	S_C
BPB_RsvdSecCnt	E	2	Số sector để dành (khác 0) (Số sector trước bảng FAT)	S_B
BPB_NumFATs	10	1	Số bảng FAT	N_F
BPB_RootEntCnt	11	2	FAT12, FAT16: số entry trong bảng RDET FAT32: có giá trị là 0	N_{RDET}
BPB_TotSec16	13	2	FAT12, FAT16: tổng số sector của Volume FAT32: có giá trị là 0	S_V
BPB_Media	15	1	Loại Voulme	
BPB_FATSz16	16	2	FAT12, FAT16: số sector trong 1 bảng FAT FAT32: có giá trị là 0 (BPB_FATSz32)	S_F
BPB_SecPerTrk	18	2	Số sectors/track	
BPB_NumHeads	1A	2	Số heads	
BPB_HiddSec	1C	4	Số sector ẩn trước Volume	
BPB_TotSec32	20	4	Số sector trong Volume. Nếu bằng 0, BPB_TotSec16 phải khác 0	N_V

Cấu trúc 36 bytes đầu tiên trong Bootsector

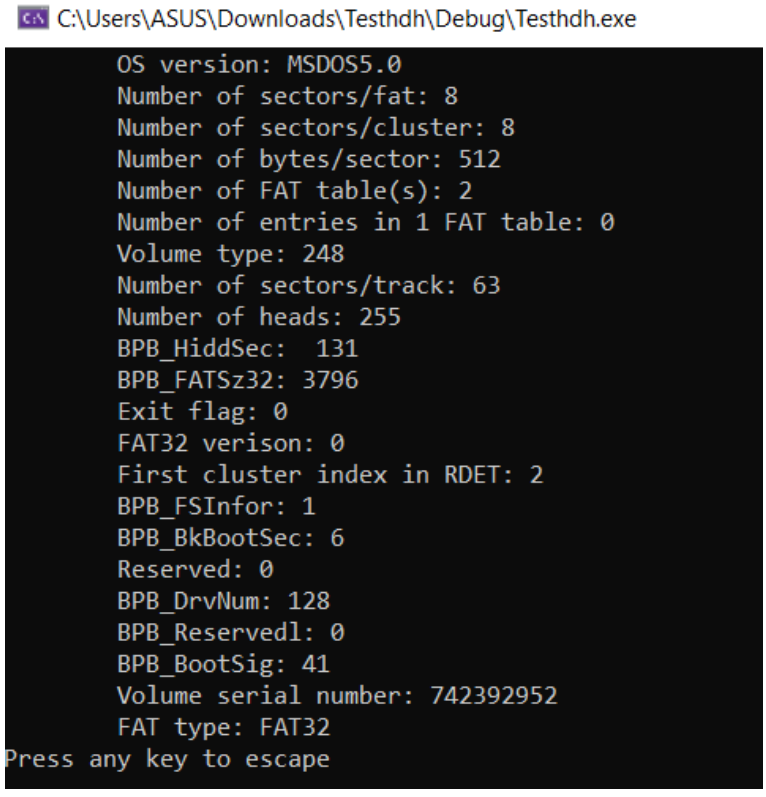
BOOTSECTOR – CẤU TRÚC

Name	Offset hexa	Size (bytes)	Description
BPB_FATSz32	**24**	**4**	**số sector trong 1 bảng FAT** **BPB_FATSz16 must be 0.**
BPB_ExtFlags	28	2	0-3: chỉ số bảng FAT active Bits 4-6: dành riêng 7: 0 – cập nhật lên tất cả các bảng FAT 1 – chỉ cập nhật lên bảng FAT active 8-15: dành riêng
BPB_FSVer	2A	2	Version FAT32 (byte thấp mirror)
BPB_RootClus	**2C**	**4**	**Chỉ số cluster đầu tiên của RDET (thông thường: 2)**
BPB_FSInfo	30	2	Chỉ số sector chứa FSINFO – thông tin sector trống. (thông thường: 1)
BPB_BkBootSec	32	2	Chỉ số sector chứa bản sao của bootsector (thông thường: 6)
BPB_Reserved	34	12	Dành riêng
BS_DrvNum	**40**	**1**	**Ký hiệu vật lý đĩa (0x00: floppy disks, 0x80: hard disks).**
BS_Reserved1	41	1	Dành riêng
BS_BootSig	42	1	Ký hiệu nhận diện HĐH (0x29).
BS_VolID	43	4	Volume serial number..
BS_VolLab	47	11	Volume label.
BS_FilSysType	52	8	Chuỗi nhận diện loại FAT: "**FAT32** ".
	5A	420	Boot code
	1FE	2	Dấu hiệu kết thúc bootsector (0x55AA)

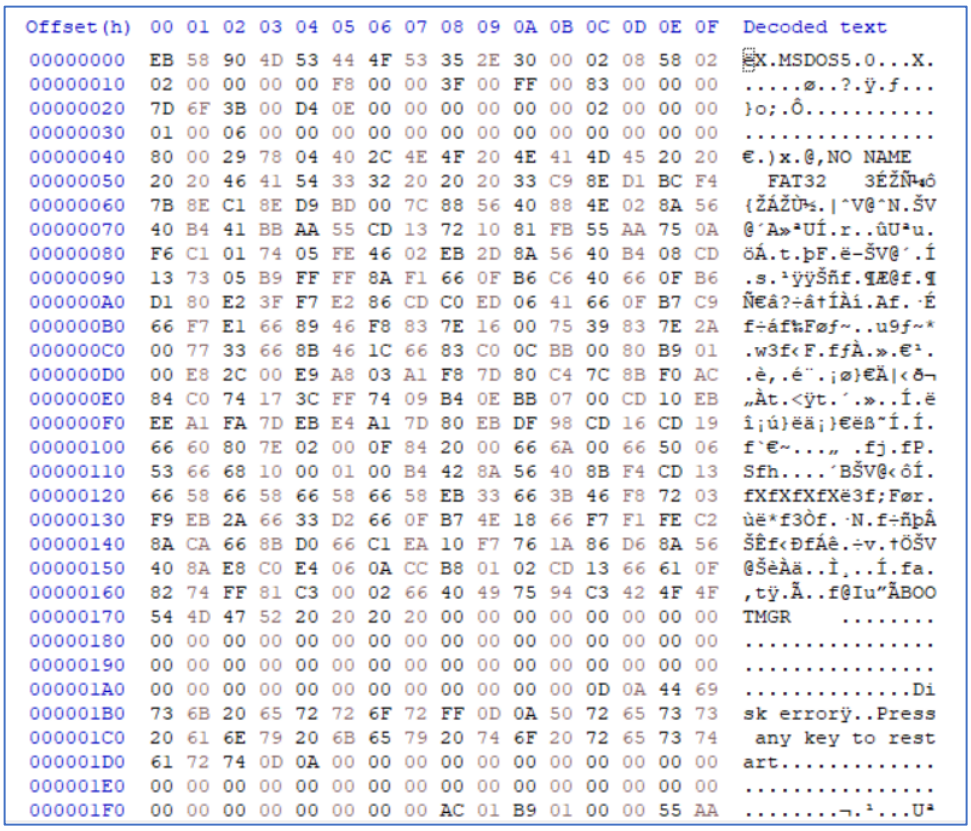
9

FAT32: Cấu trúc 476 bytes còn lại trong Bootsector

- Thông tin xuất ra bao gồm những thông tin có trong bảng trên



- Để kiểm tra thông tin của ổ đĩa đã đọc được đúng hay chưa nhóm em sử dụng **disk editor** để kiểm tra:



Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	EB	58	90	4D	53	44	4F	53	35	2E	30	00	02	08	58	02	ëX.MSDOS5.0...X.
00000010	02	00	00	00	00	F8	00	00	3F	00	FF	00	83	00	00	00	.....ø..?.ÿ.ƒ...
00000020	7D	6F	3B	00	D4	0E	00	00	00	00	00	00	02	00	00	00	}o;.Ô..........
00000030	01	00	06	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000040	80	00	29	78	04	40	2C	4E	4F	20	4E	41	4D	45	20	20	€.)x.@,NO NAME
00000050	20	20	46	41	54	33	32	20	20	20	33	C9	8E	D1	BC	F4	FAT32 3ÉŽÑ¼ô
00000060	7B	8E	C1	8E	D9	BD	00	7C	88	56	40	88	4E	02	8A	56	{ŽÁŽÙ½.|ˆV@ˆN.ŠV
00000070	40	B4	41	BB	AA	55	CD	13	72	10	81	FB	55	AA	75	0A	@´A»ªUÍ.r..ûUªu.
00000080	F6	C1	01	74	05	FE	46	02	EB	2D	8A	56	40	B4	08	CD	öÁ.t.þF.ë-ŠV@´.Í
00000090	13	73	05	B9	FF	FF	8A	F1	66	0F	B6	C6	40	66	0F	B6	.s.¹ÿÿŠñf.¶Æ@f.¶
000000A0	D1	80	E2	3F	F7	E2	86	CD	C0	ED	06	41	66	0F	B7	C9	Ñ€â?÷â†ÍÀí.Af.·É
000000B0	66	F7	E1	66	89	46	F8	83	7E	16	00	75	39	83	7E	2A	f÷áf‰Føƒ~..u9ƒ~*
000000C0	00	77	33	66	8B	46	1C	66	83	C0	0C	BB	00	80	B9	01	.w3f‹F.fƒÀ.».€¹.
000000D0	00	E8	2C	00	E9	A8	03	A1	F8	7D	80	C4	7C	8B	F0	AC	.è,.é¨.¡ø}€Ä|‹ð¬
000000E0	84	C0	74	17	3C	FF	74	09	B4	0E	BB	07	00	CD	10	EB	„Àt.<ÿt.´.».Í.ë
000000F0	EE	A1	FA	7D	EB	E4	A1	7D	80	EB	DF	98	CD	16	CD	19	î¡ú}ëä¡}€ëß˜Í.Í.
00000100	66	60	80	7E	02	00	0F	84	20	00	66	6A	00	66	50	06	f`€~...„ .fj.fP.
00000110	53	66	68	10	00	01	00	B4	42	8A	56	40	8B	F4	CD	13	Sfh....´BŠV@‹ôÍ.
00000120	66	58	66	58	66	58	66	58	EB	33	66	3B	46	F8	72	03	fXfXfXfXë3f;Før.
00000130	F9	EB	2A	66	33	D2	66	0F	B7	4E	18	66	F7	F1	FE	C2	ùë*f3Òf.·N.f÷ñþÂ
00000140	8A	CA	66	8B	D0	66	C1	EA	10	F7	76	1A	86	D6	8A	56	ŠÊf‹ÐfÁê.÷v.†ÖŠV
00000150	40	8A	E8	C0	E4	06	0A	CC	B8	01	02	CD	13	66	61	0F	@ŠèÀä..Ì¸..Í.fa.
00000160	82	74	FF	81	C3	00	02	66	40	49	75	94	C3	42	4F	4F	‚tÿ.Ã..f@Iu”ÃBOO
00000170	54	4D	47	52	20	20	20	20	00	00	00	00	00	00	00	00	TMGR ....
00000180	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000001A0	00	00	00	00	00	00	00	00	00	00	00	00	0D	0A	44	69	..............Di
000001B0	73	6B	20	65	72	72	6F	72	FF	0D	0A	50	72	65	73	73	sk errorÿ..Press
000001C0	20	61	6E	79	20	6B	65	79	20	74	6F	20	72	65	73	74	any key to rest
000001D0	61	72	74	0D	0A	00	00	00	00	00	00	00	00	00	00	00	art.............
000001E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000001F0	00	00	00	00	00	00	00	00	AC	01	B9	01	00	00	55	AA	........¬.¹...Uª

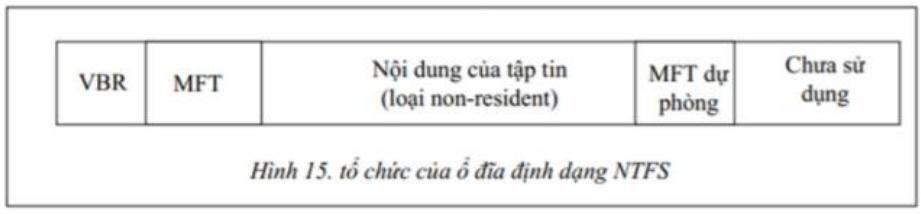
3. ĐỌC BOOTSECTOR CỦA NTFS

- Hàm **readBootSectorNTFS**: Xuất ra những thông tin cần lấy của ổ đĩa NTFS dưới dạng số nguyên cần lấy hoặc kiểu ASCII bằng cách gọi hàm **sectorToDec** và truyền vào sector bắt đầu và số lượng byte cần đọc.

```
115  void readBootSectorNTFS(BYTE sector[512])
116  {
117      cout << "\tSystem ID:  ";
118      for (int i = 3; i < 11; i++) cout << char(int(sector[i]));
119      cout << "\n\tSo byte cua Sector: " << dec << sectorToDec(sector, 11, 2);
120      cout << "\n\tSo sector cua cluster: " << dec << sectorToDec(sector, 13, 2);
121      cout << "\n\tLoai dia: " << hex << setfill('0') << setw(2) << int(sector[21]);
122      cout << "\n\tSo sector cua volume: " << dec << sectorToDec(sector, 40, 3);
123      cout << "\n\tVi tri MFT: " << sectorToDec(sector, 48, 8);
124      cout << "\n\tVi tri ban sao MFT: " << sectorToDec(sector, 56, 8);
125      cout << "\n\tSo cluster cua MFT record: " << sectorToDec(sector, 64, 1);
126  }
```

- Tương tự như hàm readBootSectorFat32 khi đọc fat 32. Hàm này cũng lấy chỉ số index dựa vào công thức **index = row * 16 + col** (row: Vị trí dòng , col : Vị trí cột của offset cần lấy)
- Vị trí của các offset của NTFS sẽ có những điểm giống và khác của Fat32 ở một số chỗ sẽ thể hiện ở 2 bảng trong phần đọc bootsector của Fat32 và NTFS.

Tổ chức của ổ đĩa logic định dạng NTFS được minh họa ở Hình 15.



Hình 15. tổ chức của ổ đĩa định dạng NTFS

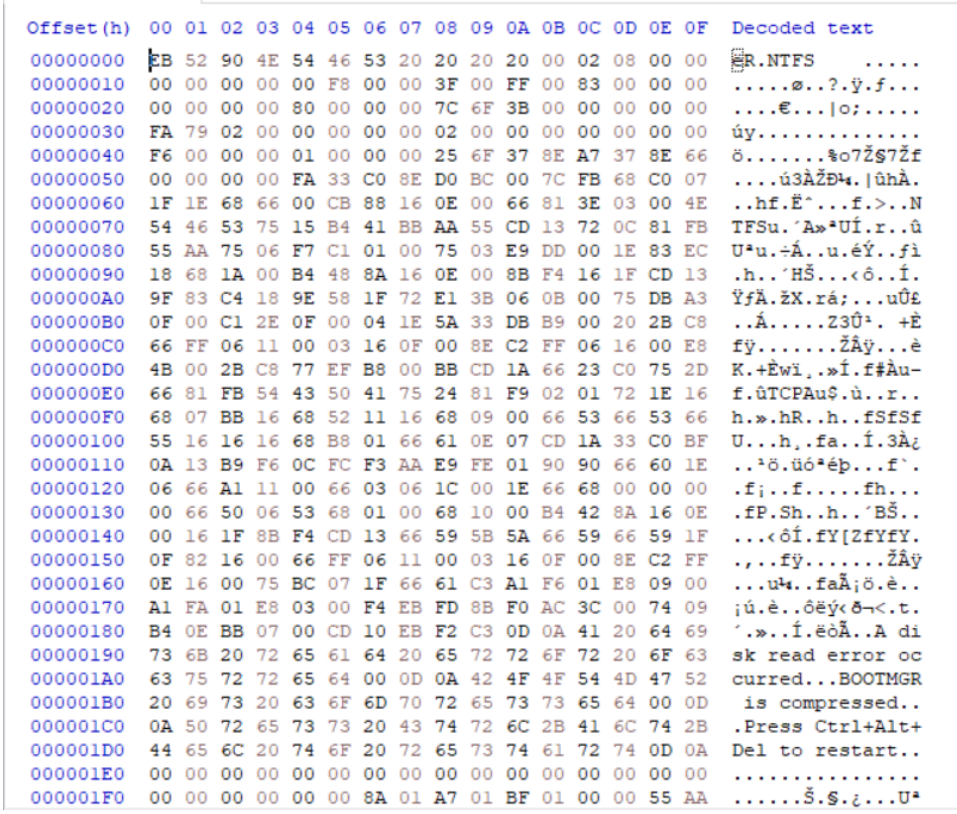
- **Volume Boot Record**, thường được gọi là **Partition Boot Sector**, một loại bootsector, được lưu trữ trên một phân vùng cụ thể trên ổ cứng hoặc thiết bị lưu trữ khác, có chứa code máy tính cần thiết để bắt đầu quá trình boot. VBR chứa: mã khởi động, BPB, thông báo lỗi, và một số thông tin khác.
- Để đọc thông tin về bootsector của NTFS ta sẽ đọc lần lượt các byte mô tả các trường trong **BPB** (Bios Parameter Block) bắt đầu tại offset 0xB tới offset 0x53 trong VBR, có kích thước 73 byte. BPB chứa một số thông tin mô tả về tổ chức của một ổ đĩa logic và hệ thống quản lý tập tin. Ví dụ: kích thước của một sector, số sector trong một cluster, tổng số sector trong ổ đĩa logic, vị trí bắt đầu của vùng MFT… Trong trường này có nhiều thông tin nhưng những thông tin quan trọng được gói gọn lại trong bảng dưới đây

Offset (hex)	Số byte	Nội dung
B	2	Số byte của Sector (thường là 512)
D	2	Số sector của Cluster
15	1	Loại đĩa (thường là F8 - đĩa cứng)
28	8	Số sector của Volume
30	8	Vị trí MFT – tính theo chỉ số cluster vật lý
38	8	Vị trí bản sao MFT (chỉ số cluster vật lý)
40	1	Số cluster của MFT record (=2^{-K} nếu K âm)
44	1	Số cluster của Index Buffer
48	8	Serial Number

- Thông tin xuất ra khi đọc ổ đĩa NTFS bao gồm những thông tin có trong bảng trên:

```
C:\Users\ASUS\Downloads\Testhdh\Debug\Testhdh.exe
Enter NTFS drive (E/G/...): F
        System ID:  NTFS
        So byte cua Sector: 512
        So sector cua cluster: 8
        Loai dia: f8
        So sector cua volume: 3895164
        Vi tri MFT: 162298
        Vi tri ban sao MFT: 2
        So cluster cua MFT record: 246
Press any key to escape
```

- Tương tự để kiểm tra thông tin ổ đĩa đọc đúng hay không nhóm cũng sử dụng phần mềm disk editor:



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	EB	52	90	4E	54	46	53	20	20	20	20	00	02	08	00	00	ëR.NTFS ....
00000010	00	00	00	00	00	F8	00	00	3F	00	FF	00	83	00	00	00	.....ø..?.ÿ.ƒ...
00000020	00	00	00	00	80	00	00	00	7C	6F	3B	00	00	00	00	00	....€...|o;.....
00000030	FA	79	02	00	00	00	00	00	02	00	00	00	00	00	00	00	úy..............
00000040	F6	00	00	00	01	00	00	00	25	6F	37	8E	A7	37	8E	66	ö.......%o7Ž§7Žf
00000050	00	00	00	00	FA	33	C0	8E	D0	BC	00	7C	FB	68	C0	07	....ú3ÀŽÐ¼.|ûhÀ.
00000060	1F	1E	68	66	00	CB	88	16	0E	00	66	81	3E	03	00	4E	..hf.Ë^...f.>..N
00000070	54	46	53	75	15	B4	41	BB	AA	55	CD	13	72	0C	81	FB	TFSu.´A»ªUÍ.r..û
00000080	55	AA	75	06	F7	C1	01	00	75	03	E9	DD	00	1E	83	EC	Uªu.÷Á..u.éÝ..ƒì
00000090	18	68	1A	00	B4	48	8A	16	0E	00	8B	F4	16	1F	CD	13	.h..´HŠ...‹ô..Í.
000000A0	9F	83	C4	18	9E	58	1F	72	E1	3B	06	0B	00	75	DB	A3	ŸƒÄ.žX.rá;...uÛ£
000000B0	0F	00	C1	2E	0F	00	04	1E	5A	33	DB	B9	00	20	2B	C8	..Á.....Z3Û¹. +È
000000C0	66	FF	06	11	00	03	16	0F	00	8E	C2	FF	06	16	00	E8	fÿ.......ŽÂÿ...è
000000D0	4B	00	2B	C8	77	EF	B8	00	BB	CD	1A	66	23	C0	75	2D	K.+Èwï¸.»Í.f#Àu-
000000E0	66	81	FB	54	43	50	41	75	24	81	F9	02	01	72	1E	16	f.ûTCPAu$.ù..r..
000000F0	68	07	BB	16	68	52	11	16	68	09	00	66	53	66	53	66	h.».hR..h..fSfSf
00000100	55	16	16	16	68	B8	01	66	61	0E	07	CD	1A	33	C0	BF	U...h¸.fa..Í.3À¿
00000110	0A	13	B9	F6	0C	FC	F3	AA	E9	FE	01	90	90	66	60	1E	..¹ö.üóªéþ...f`.
00000120	06	66	A1	11	00	66	03	06	1C	00	1E	66	68	00	00	00	.f¡..f.....fh...
00000130	00	66	50	06	53	68	01	00	68	10	00	B4	42	8A	16	0E	.fP.Sh..h..´BŠ..
00000140	00	16	1F	8B	F4	CD	13	66	59	5B	5A	66	59	66	59	1F	...‹ôÍ.fY[ZfYfY.
00000150	0F	82	16	00	66	FF	06	11	00	03	16	0F	00	8E	C2	FF	.‚..fÿ.......ŽÂÿ
00000160	0E	16	00	75	BC	07	1F	66	61	C3	A1	F6	01	E8	09	00	...u¼..faÃ¡ö.è..
00000170	A1	FA	01	E8	03	00	F4	EB	FD	8B	F0	AC	3C	00	74	09	¡ú.è..ôëý‹ð¬<.t.
00000180	B4	0E	BB	07	00	CD	10	EB	F2	C3	0D	0A	41	20	64	69	´.»..Í.ëòÃ..A di
00000190	73	6B	20	72	65	61	64	20	65	72	72	6F	72	20	6F	63	sk read error oc
000001A0	63	75	72	72	65	64	00	0D	0A	42	4F	4F	54	4D	47	52	curred...BOOTMGR
000001B0	20	69	73	20	63	6F	6D	70	72	65	73	73	65	64	00	0D	is compressed..
000001C0	0A	50	72	65	73	73	20	43	74	72	6C	2B	41	6C	74	2B	.Press Ctrl+Alt+
000001D0	44	65	6C	20	74	6F	20	72	65	73	74	61	72	74	0D	0A	Del to restart..
000001E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000001F0	00	00	00	00	00	00	8A	01	A7	01	BF	01	00	00	55	AA	......Š.§.¿...Uª

III. HIỂN THỊ THÔNG TIN CÂY THƯ MỤC CỦA PHÂN VÙNG

- 1 số biến viết tắt được tính trong hàm main và được sử dụng để truyền cứng vào 1 vài hàm ở phần này. Như đã trình bày phần II.2, để truy xuất index trong mảng 1 chiều **bootSector**, ta dùng công thức index = row * 16 + col
    - **Sb**: số sector vùng boot sector, 0E - 2 byte tương ứng 2 index là 14, 15
    - **Nf**: số bảng FAT, 10 - 1 byte tương ứng index = 1*16 + 0 = 16
    - **Sc**: số sector cho 1 cluster, 0D - 1 byte tương ứng index = 13

- **Sr**: số sector cho RDET, vì là FAT32 nên Sr = 0
- **Sf**: số sector cho 1 bảng FAT, 16 – 2 byte tương ứng 2 index là 22, 23. Nếu giá trị tại 2 index đó bằng 0 thì đi tới 24 – 4 byte tương ứng 4 index 36, 37, 38, 39

```
48              //các thông số quan trọng
49              int Sb = bootSector[15] << 8 | bootSector[14];  //sector vùng bootsector
50              int Nf = bootSector[16];                        // số bảng FAT
51              int Sc = bootSector[13];                        //số sector cho 1 cluster
52              int Sr = 0;                                     //Fat32 -> Sr=0
53              int Sf = bootSector[23] << 8 | bootSector[22];  //số sector cho 1 bảng FAT, khởi tạo là xx16 - 2 byte
54              if (Sf == 0x00) { //xx24 - 4 byte
55                  Sf = bootSector[39] << 8 | bootSector[38];
56                  Sf = Sf << 8 | bootSector[37];
57                  Sf = Sf << 8 | bootSector[36];
58              }
```

1. ĐỊNH DẠNG KIỂU DỮ LIỆU ĐỂ PHỤC VỤ CHO ĐỌC THÔNG TIN CÂY THƯ MỤC

- Gồm 2 struct sẽ sử dụng xuyên suốt cho quá trình đọc thông tin cây thư mục: **Root** và **FEntity**

```
struct Root {
    BYTE fileName[8];
    BYTE extension[3];
    BYTE fileAttributes;
    BYTE reserved;
    BYTE createTime_ms;
    //---nếu là entry phụ: những thuộc tính dưới đây nằm trong file name
    BYTE createTime[2];
    BYTE createDate[2];
    BYTE accessedDate[2];
    BYTE highCluster[2];
    BYTE modifiedTime[2];
    BYTE modifiedDate[2];
    //---
    BYTE lowCluster[2];
    //---nếu là entry phụ: thì đây là 4 ký tự trong file name
    BYTE sizeofFile[4];
};

struct FEntity {
    vector<BYTE> name;
    BYTE attribute;
    int startingCluster;
    vector<int> claimedClusters;
    vector<int> claimedSectors;
    int fsize; // bytes
    vector<BYTE> data;
};
```

- Struct **Root**: chứa các thuộc tính của 1 entry (32 byte) trong RDET hoặc SDET, struct này có thể được coi là struct "cấp thấp" vì nó sẽ hỗ trợ cho struct Fentity. Ý nghĩa các thuộc tính được thể hiện ở tên thuộc tính (Ví dụ: **fileName**: tên file/folder)
- Struct **FEntity**: là struct theo hướng interface nhiều hơn, sau khi đọc dữ liệu entry vào 1 thể hiện Root (Root instance), thì struct **FEntity** có nhiệm vụ chuyển đổi (ví dụ: thuộc tính **startingCluster**) hoặc gộp (ví dụ: thuộc tính **name**) 1 số thuộc tính từ thể hiện Root này để dễ dàng cho các tác vụ truy xuất cũng như in ra màn hình. Ý nghĩa các thuộc tính:
    - **name**: tên của file/folder
    - **attribute**: trạng thái thuộc tính của file
    - **startingCluster**: chỉ số cluster bắt đầu
    - **claimedClusters**: chỉ số (các) cluster chiếm giữ
    - **claimedSectors**: chỉ số các sector lưu trữ trên đĩa cứng
    - **fsize**: kích thước file/folder, tính theo đơn vị byte
    - **data**: nội dung tập tin (nếu file có đuôi mở rộng là txt)

2. ĐỌC VÀ HIỂN THỊ CÂY THƯ MỤC (RDET) CỦA FAT32

a) Các hàm hỗ trợ cho phần này

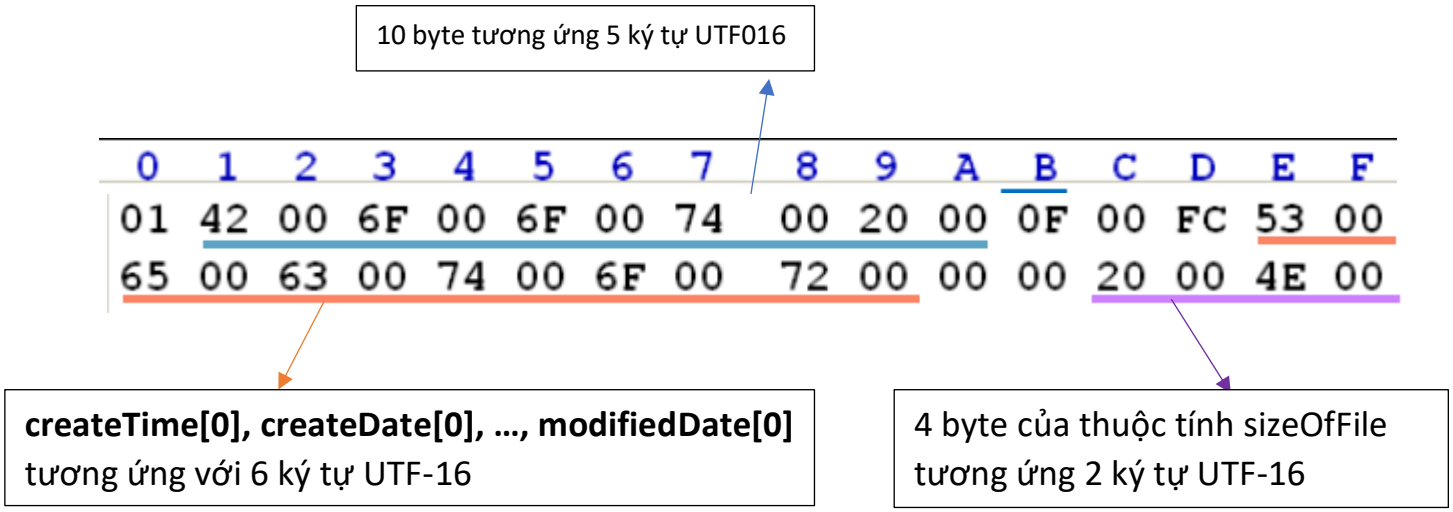
- Hàm đọc tên file: có chức năng đọc dữ liệu tên file/folder từ entry chính hoặc entry phụ cho thể hiện **FEntity f**

`void readFileName(Root root, FEntity& f, int mode)`	
`Root root`	Thông tin entry cần khai thác tên
`FEntity& f`	Thể hiện FEntity cần lưu tên
`int mode`	mode = 0 (default) → entry **root** ở trên truyền vào là entry chính; ngược lại mode = 1

```
190     //mode=0 -> entry chính, mode=1 -> entry phụ
191     void readFileName(Root root, FEntity& f, int mode) {
192         if (mode == 0) {
193             for (int i = 0; i < 8; ++i)
194                 f.name.push_back(root.fileName[i]);
195
196             if (root.extension[0] != 0x20) {
197                 f.name.push_back('.');
198                 for (int i = 0; i < 3; i++) {
199                     f.name.push_back(root.extension[i]);
200                 }
201             }
202         }
203         else {
204             for (int i = 1; i < 8; ++i) {
205                 f.name.push_back(root.fileName[i]);
206             }
207
208             for (int i = 0; i < 3; ++i) {
209                 f.name.push_back(root.extension[i]);
210             }
211
212             f.name.push_back(root.createTime[0]);
213             f.name.push_back(root.createDate[0]);
214             f.name.push_back(root.accessedDate[0]);
215             f.name.push_back(root.highCluster[0]);
216             f.name.push_back(root.modifiedTime[0]);
217             f.name.push_back(root.modifiedDate[0]);
218
219             for (int i = 0; i < 4; ++i) {
220                 f.name.push_back(root.sizeofFile[i]);
221             }
222         }
223     }
```

- Đối với entry truyền vào là entry chính (mode=0), thì tên file/folder luôn có tối đa 8 ký tự nên vòng for đầu tiên dòng 193 dùng để lưu tên. Để đọc tiếp đuôi file (folder không có đuôi nên điều kiện dòng 196 sẽ không được thỏa), ta sẽ khai thác thuộc tính **extension** của struct Root, vì đuôi file có tối đa 3 ký tự nên vòng for dòng 198 dùng để lưu 3 ký tự ấy
- Đối với entry truyền vào là entry phụ (mode=1) thì thuộc tính **fileName** của biến root có byte đầu tiên thể hiện số thứ tự entry → không cần quan tâm vì không chứa tên. Chính vì thế vòng for ở dòng 204 có chỉ số i bắt đầu từ 1 thay vì 0). Ta chỉ quan tâm 10 byte tiếp theo (tức 5 ký tự UTF-16) sau byte đầu tiên vì 10 byte này chứa ký tự trong tên file. Sau khi thực hiện xong vòng for dòng 208 thì 10 byte đầu được đọc thành công. Ngoài ra, các ký tự tên file còn nằm ẩn ở các thuộc tính **createTime[0], createDate[0], …, modifiedDate[0]** tương ứng với 6 ký tự UTF-16 (như struct Root cũng có chỉ ra ở phần comment), và nằm ở 4 byte của thuộc tính sizeOfFile tương ứng 2 ký tự UTF-16 (vòng for dòng 219)

Ví dụ minh họa cho cách hàm hoạt động trên entry phụ:



- Kết thúc hàm, thuộc tính **name** của biến tham chiếu **f** sẽ được cập nhật

- Hàm đọc bảng FAT32: có chức năng đọc thông tin bảng FAT32 và trả ra vector<DWORD> với mỗi phần tử trong mảng là phần tử FAT32 4 byte

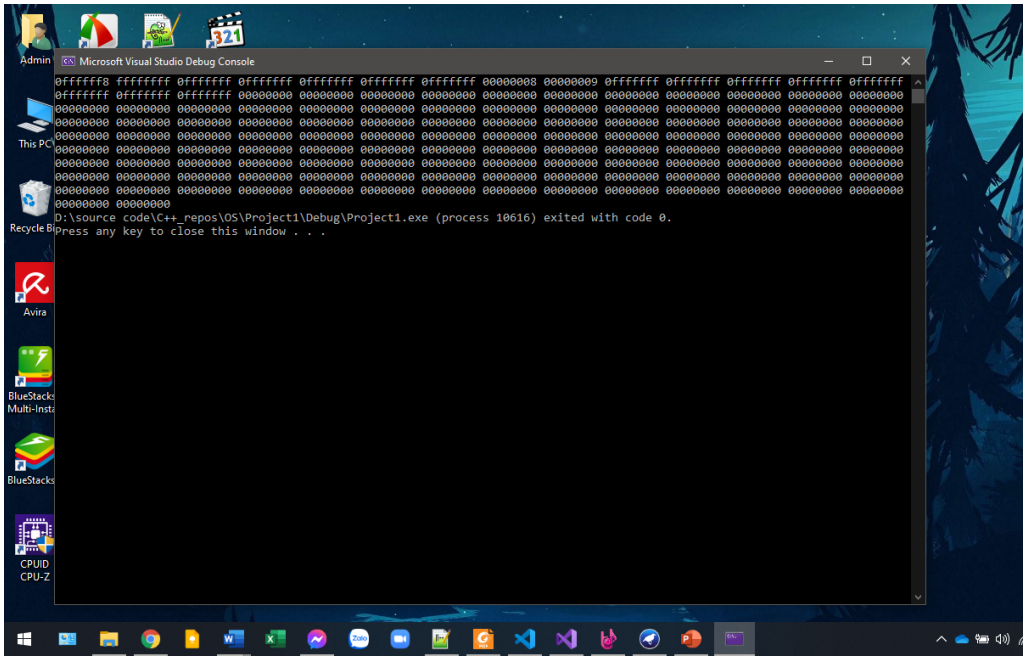
`vector<DWORD> readFat(BYTE bootSector[512], HANDLE fat32_disk)`	
`BYTE bootSector[512]`	Thông tin (hay bảng) bootSector
`HANDLE fat32_disk`	Thông tin của ổ đĩa FAT 32, có kiểu là HANDLE

```
256 vector<DWORD> readFat(BYTE bootSector[512], HANDLE fat32_disk) {
257     int fatPos = bootSector[15] << 8 | bootSector[14]; //Sb, xxxE - 2 byte
258     DWORD dwFilePointer = SetFilePointer(fat32_disk, (512 * fatPos), NULL, FILE_BEGIN);
259     BYTE* byteRoot = new BYTE[512];
260     DWORD dwBytesRead;
261     vector<DWORD> fat;
262
263     if (!ReadFile(fat32_disk, byteRoot, 512, &dwBytesRead, NULL))
264         cout << "Error in Reading FAT32 disk.\n";
265     else {
266         for (int i = 0; i < 512; ++i) {
267             DWORD group = 0;
268             for (int j = i + 3; j >= i; --j)
269                 group = group << 8 | byteRoot[j];
270             i += 3;
271             fat.push_back(group);
272         }
273     }
274     return fat;
275 }
```

- Đầu tiên cần xác định vị trí đọc của bảng FAT, đó là vị trí 000E - 2 byte trong bảng boot sector tương ứng vị trí 14, 15 trong mảng **bootSector**. Ta cần đọc ngược (cách đọc ngược

được trình bày ở các ý sau) 2 byte này vào biến **fatPos** để ra vị trí chính xác cần nhảy đến. Và vì hàm **SetFilePointer** có tham số 2 tính theo đơn vị byte nên ta cần nhân **fatPos** với 512 để ra vị trí đọc tính theo byte

- Tiếp theo, tại dòng 263 đọc 512 byte bảng FAT vào mảng **byteRoot**
- Ở vòng for dòng 206, vì mỗi phần tử FAT32 chiếm 4 byte nên lặp: cứ mỗi 4 byte trong mảng **byteRoot** thì sẽ đọc ngược 4 byte này → tạo thành 1 phần tử FAT32 → lưu vào biến **group**. Cuối cùng đẩy phần tử FAT32 vừa tạo thành vào mảng **fat**
- Output khi in mỗi phần tử trong bảng FAT:



b) Đọc cây thư mục gốc

❖ Giải thích hàm và các tham số truyền vào

`vector<FEntity> readXdet(BYTE bootSector[512], HANDLE fat32_disk, vector<DWORD> fat, int xdetPos, vector<int> info)`	
`BYTE bootSector[512]`	Thông tin (hay bảng) bootSector
`HANDLE fat32_disk`	Thông tin của ổ đĩa FAT 32, có kiểu là HANDLE
`vector<DWORD> fat`	Bảng FAT, với mỗi phần tử trong mảng là 1 phần tử FAT32
`int xdetPos`	Vị trí sector bắt đầu của RDET hoặc SDET
`vector<int> info`	Là thông số cơ bản đọc từ RDET gồm **đúng** các phần tử { Sb, Nf, Sf, Sr, Sc}

- Trả về: 1 danh sách (mảng) gồm các thực thể file/folder trong RDET

- Hàm này dùng cho cả 2 việc: đọc RDET và SDET. Chính vì thế hàm mới có tên là **readXdet** với **Xdet** là Rdet hoặc Sdet

❖ Mô tả bước thực hiện:

- Khai báo các biến để dùng, trong có có 1 số biến quan trọng sau
  - **res**: kết quả trả về
  - **dwFilePointer**: là con trỏ đọc, dùng để đọc từ 1 vị trí trong đĩa FAT32
  - **stRoot**: biến entry kiểu Root dùng để lưu thông tin của 1 entry đọc từ RDET
  - **byteRoot**: biến entry kiểu BYTE dùng lưu thông tin dạng byte của 1 entry đọc từ RDET, khởi tạo mảng gồm các phần tử có giá trị 0
  - **Sb, Nf, Sf, Sr, Sc:** cần cho việc đọc và lưu chỉ số sector lưu trữ trên đĩa cứng

```
283  vector<FEntity> readXdet(BYTE bootSector[512], HANDLE fat32_disk, vector<DWORD> fat, int xdetPos, vector<int> info) {
284      vector<FEntity> res;
285      DWORD dwFilePointer;
286      DWORD dwBytesRead;
287      Root stRoot;
288      BYTE byteRoot[512];
289      memset(&byteRoot, 0, 512);
290
291      // info = {Sb, Nf, Sf, Sr, Sc}
292      int Sb = info[0], Nf = info[1], Sf = info[2], Sr = info[3], Sc = info[4];
```

- Một số tiền điều kiện cần kiểm tra trước khi thực hiện nhiệm vụ chính là đọc RDET:
  - Sau khi kiểm tra ổ FAT32 có khác rỗng ở dòng 291 hay không, nếu thỏa mãn thì tiếp tục thiết lập vị trí đọc RDET trong FAT32 nhờ hàm SetFilePointer. Vì tham số thứ 2 của hàm SetFilePointer là 1 số nguyên thể hiện số byte đọc, thế nên cần chuyển **xdetPos** (hay trong trường hợp này là rdetPos) từ đơn vị là chỉ số sector sang đơn vị byte bằng cách nhân 512. Vì vậy nên **(512 * xdetPos)** sẽ được truyền vào tham số thứ 2 của hàm SetFilePointer
  - Nếu **xdetPos** là 1 giá trị thể hiện đúng chỉ số sector đầu tiên của bảng RDET, nghĩa là xdetPos = Sb + Nf*Sf thì lệnh if ở dòng 294 sẽ được thỏa mãn và đi vào hàm xử lý tiếp, ngược lại sẽ in thông báo lỗi "INVALID_SET_FILE_POINTER" ra màn hình
  - Vì 1 lần đọc sẽ đọc 512 byte, nên nếu RDET có kích thước lớn hơn 512 byte sẽ cần nhiều lần đọc. Vì vậy, vòng do..while có tác dụng lặp qua các lần đọc với mỗi lần đọc thì sẽ đọc 512 byte. Với điều kiện của mỗi lần lặp là cần phải đọc vào ổ đĩa 512 byte tiếp theo thành công (dòng 297)
- Biến bool **bNoEntry** được khai báo có tác dụng check entry có trống (0x00) hay không, nếu có thì biến này bật true thể hiện là đã đọc chạm tới entry trống, khi đó break vòng do..while ở dưới để kết thúc. Biến này sẽ được làm rõ hơn trong các ý dưới

- Xét 1 lần lặp 512 byte trong vòng do..while, ta sẽ tiếp tục khai thác với mỗi entry tương ứng 32 byte thì có trường nào được khai thác (vòng for dòng 303)

```
291     if (fat32_disk != NULL) {
292         dwFilePointer = SetFilePointer(fat32_disk, (512 * xdetPos), NULL, FILE_BEGIN);
293
294         if (dwFilePointer != INVALID_SET_FILE_POINTER) {
295             BOOL bNoEntry = FALSE;
296             do {
297                 if (!ReadFile(fat32_disk, byteRoot, 512, &dwBytesRead, NULL))
298                     cout << "Error in Reading Root Entry.\n";
299                 else {
300                     BYTE* pByteRoot = byteRoot;
301
302                     // đọc mỗi entry (1 entry = 32 byte)
303                     for (int i = 0; i < (512 / 32); i++) { ... }
396                 }
397                 if (bNoEntry)
398                     break;
399             } while (true);
400         }
401         else
402             cout << "INVALID_SET_FILE_POINTER";
403     }
404     else
405         cout << "cant open handle";
406     return res;
407 }
```

- Thực thi nhiệm vụ chính: đọc và lấy thông tin mỗi entry:
  - Mỗi entry 32 byte được đại diện bằng 1 biến **f** và 1 biến stRoot. Vì mỗi entry = 32 byte nên ta lần lượt copy 32 byte từ **pByteRoot** (là mảng tạm thời dùng lưu 512 byte từ **byteRoot**) bằng lệnh memcpy. Ngoài ra cần biến bool **hasSubEntry** để biết nếu entry đang xét có entry phụ (true) hay không (false).
  - Đầu tiên, kiểm tra trạng thái của mỗi entry. Ở đây, có 2 trạng thái quan trọng nhóm em muốn kiểm tra: trạng thái entry trống (byte đầu là 0x00, dòng 309) và trạng thái entry bị xóa (byte đầu là 0xE5, dòng 314). Nếu là trạng thái trống, tức là ta đã đọc hết dữ liệu cần đọc trong RDET nên gán **bNoEntry = true** để break ra khỏi do..while, ngừng việc đọc RDET. Nếu là trạng thái bị xóa, thì file/folder đó tụi em không quan tâm (tức không hiển thị lên console) cho nên **continue** để bỏ qua file/folder bị xóa
  - Tiếp đến, kiểm tra liệu entry đang xét có là entry phụ hay không (tức thuộc tính của entry = 0x0F?). Thấy rằng nếu có nhiều entry phụ thì ta sẽ đọc theo thứ tự từ dưới lên trên (thay vì từ trên xuống dưới như đoạn code hiện tại), vì thế nên nhóm em dùng cấu trúc dữ liệu stack để lưu trữ nhiều entry phụ theo thứ tự này. Giả sử entry đang xét là entry phụ thì ta cần phải tạm lưu lại nó vào stack rồi đọc tiếp entry tiếp theo, nếu entry tiếp theo vẫn là

entry phụ thì lại lưu vào stack,… cho đến khi đọc tới entry chính thì thoát khỏi do..while ở dòng 325. Khi thoát vòng do..while, stack hiện tại sẽ chứa (các) entry phụ của entry chính, việc cần làm tiếp theo là duyệt và đọc tên file từ mỗi entry phụ trong stack vào biến tạm **ftmp** thông qua hàm **readFileName** đã được mô tả mục a

- Vì mọi ký tự liên qua đến tên file (bao gồm tên + đuôi extension) đều đã được thể hiện thông qua tất cả entry phụ nên không cần tìm tên trong entry chính nữa. Giờ ta có thể gán **ftmp** lại vào biến **f**, khi đó tên của file/folder **f** là đầy đủ

```
302                 // đọc mỗi entry (1 entry = 32 byte)
303                 for (int i = 0; i < (512 / 32); i++) {
304                     FEntity f;
305                     memcpy(&stRoot, pByteRoot, 32);
306                     bool hasSubEntry = 0;
307
308                     // check entry có trống
309                     if (stRoot.fileName[0] == 0x00) {
310                         bNoEntry = true;
311                         break;
312                     }
313                     // check entry có bị xóa
314                     if (stRoot.fileName[0] == 0xE5) {
315                         pByteRoot += 32;
316                         continue;
317                     }
318                     //check entry phụ
319                     if (stRoot.fileAttributes == 0x0F) {
320                         stack<Root> subEntries;
321                         hasSubEntry = 1;
322                         FEntity ftmp;
323
324                         //lưu entries phụ
325                         do {
326                             subEntries.push(stRoot);
327                             pByteRoot += 32;
328                             i++;
329                             memcpy(&stRoot, pByteRoot, 32);
330                         } while (stRoot.fileAttributes == 0x0F);
331
332                         //lấy tên từ entries phụ
333                         while (!subEntries.empty()) {
334                             Root subEntry = subEntries.top();
335                             subEntries.pop();
336                             readFileName(subEntry, ftmp, 1);
337                         }
338                         f = ftmp;
339                     }
```

- Ở đoạn code phía trên, ta giả sử rằng entry có các entry phụ nên tên. Trong trường hợp nếu entry là entry chính và không có entry phụ thì biến **hasSubEntry** vẫn giữ nguyên là false và ta phải đọc tên file/folder từ entry chính như bình thường nhờ hàm **readFileName** được mô tả ở câu a. Sau khi đọc tên vào biến **f**, tiếp tục đọc thuộc tính trạng thái
- Dựa theo slide đã được cung cấp, lần lượt kiểm tra phép AND giữa thuộc tính **f.attribute** với 0x01, 0x02, …, 0x20 (các dòng if từ 347 - 357) có trả về true hay không, nếu trả về true ở dòng if nào thì thuộc tính của **f** có giá trị là số hexa của dòng if đó
- Tiếp theo, tìm chỉ số cluster bắt đầu:

- Ta có **highCluster** là phần word (2 byte) cao và cần đọc ngược lại nên dùng phép (<<) để shift **stRoot.highCluster[1]** qua trái 8 bit. Khi đó 8 bit sau cùng là 8 bit 0, ta chỉ cần cộng (|) với 8 bit của **stRoot.highCluster[0]** là đọc ngược thành công.
  Ví dụ: stRoot.highCluster = {'0x07', '0x03'} →output: 0x0307
  Giải thích: stRoot[1] = 0x03 = 0000.0000.0000.0011
  stRoot[0] = 0x07 = 0000.0000.0000.0111
  → stRoot[1] << 8 = 0000.0011.0000.0000
  → stRoot[1] << 8 | stRoot[0] = 0000.0011.0000.0111 = 0x0307

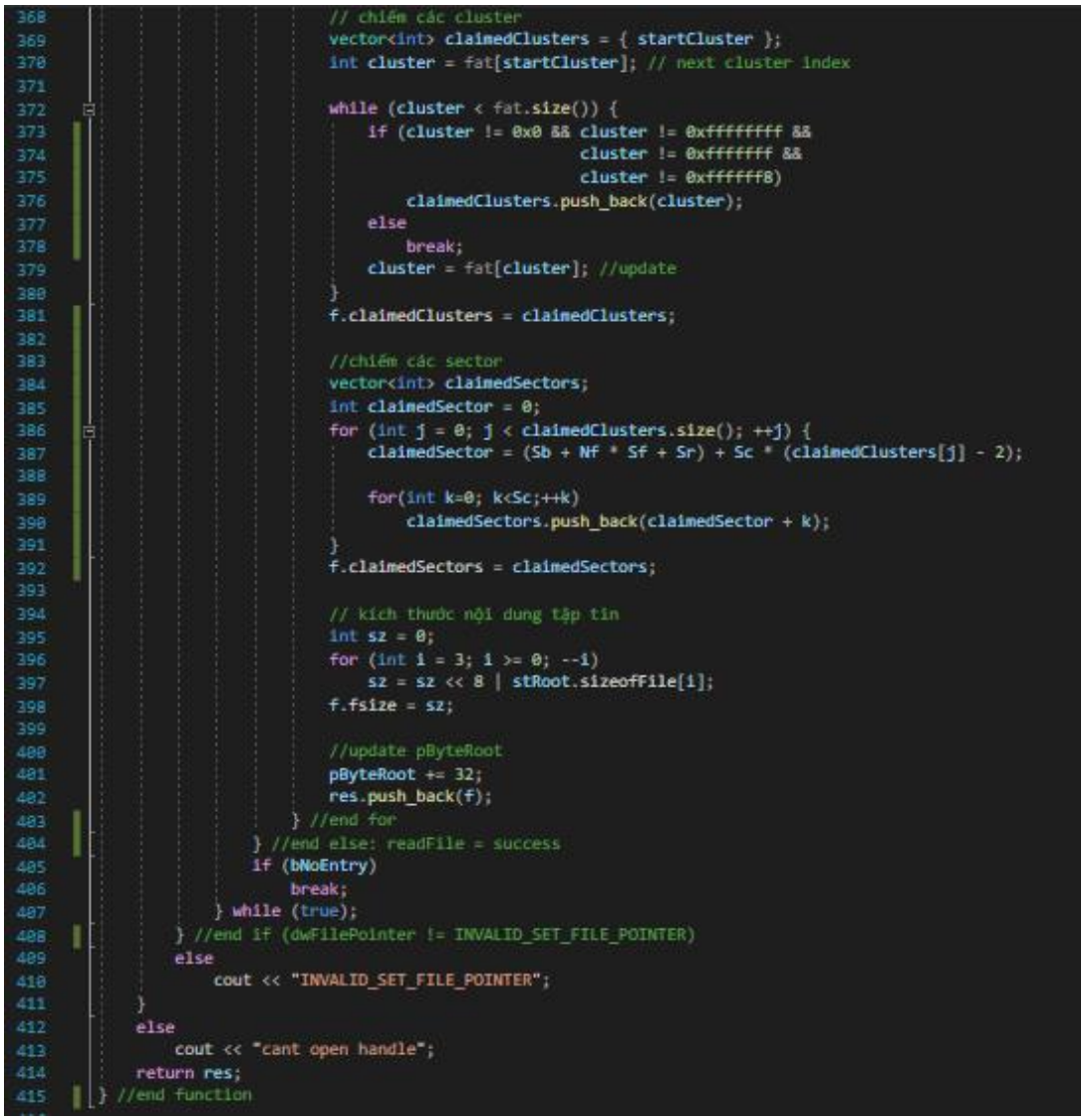
- Ta có **lowCluster** là phần word (2 byte) thấp và cần đọc ngược (đọc ngược tương tự như **highCluster**)
- Ta tính được chỉ số cluster bắt đầu **startingCluster** bằng cách nối **highCluster** và **lowCluster** theo chiều ngược lại (làm tương tự như cách đọc ngược của **highCluster**)

```
341                 //file name
342                 if (!hasSubEntry) {
343                     readFileName(stRoot, f);
344                 }
345
346                 // check thuộc tính trạng thái
347                 if (stRoot.fileAttributes & 0x01)
348                     f.attribute = 0x01;
349                 else if (stRoot.fileAttributes & 0x02)
350                     f.attribute = 0x02;
351                 else if (stRoot.fileAttributes & 0x04)
352                     f.attribute = 0x04;
353                 else if (stRoot.fileAttributes & 0x08)
354                     f.attribute = 0x08;
355                 else if (stRoot.fileAttributes & 0x10)
356                     f.attribute = 0x10;
357                 else if (stRoot.fileAttributes & 0x20)
358                     f.attribute = 0x20;
359
360                 // cluster bắt đầu
361                 int highCluster = stRoot.highCluster[1] << 8 | stRoot.highCluster[0];
362                 int  lowCluster = stRoot.lowCluster[1] << 8 | stRoot.lowCluster[0];
363                 int startCluster = (highCluster << 8 | lowCluster);
364                 f.startingCluster = startCluster;
```

- Tiếp theo, tìm các cluster mà entry đang xét chiếm giữ, các cluster được chiếm sẽ lưu vào mảng **claimedClusters**. Mảng này luôn chứa **startCluster** vì đây cũng chính là cluster chiếm giữ đầu tiên. Để tìm cluster thứ 2 (nếu có), chỉ cần truy cập vào phần tử thứ **startCluster** trong bảng **fat** (tức **fat[startCluster]**) vì các cluster trong bảng FAT được cấu trúc như 1 danh sách liên kết: tại vị trí thứ **startCluster** trong bảng FAT sẽ lưu chỉ số index của cluster tiếp theo. Từ ý tưởng này, thiết lập 1 vòng while với điều kiện là **cluster** (cũng chính là chỉ số trong bảng FAT) luôn bé hơn kích thước bảng FAT (dòng 369), lặp qua các cluster tìm được, nếu cluster đang xét rơi vào các giá trị [0x0, 0xffffffff, 0x0fffffff, 0x0ffffff8] thì đây là

điểm kết thúc → break ra khỏi vòng lặp; ngược lại thì thì tiếp tục lặp. Kết thúc vòng lặp, toàn bộ cluster chiếm giữ được lưu lại vào **f.claimedClusters**

- Tiếp theo, tìm chỉ số sector lưu trữ trên đĩa cứng. Các sector chiếm giữ sẽ được lưu trong mảng **claimedSectors**. Theo công thức được cung cấp trong slide: cluster K bất kỳ sẽ chiếm **Sc** sector bắt đầu tại sector có chỉ số (Sb + Nf * Sf + Sr) + Sc * (K- 2). Áp dụng công thức này với K là mỗi cluster chiếm giữ trong mảng **claimedClusters** để tính ra được chỉ số sector lưu trữ
- Tiếp theo, tìm kích thước nội dung tập tin, cũng chính là 4 byte cuối trong entry chính, hay **stRoot.sizeOfFile[4]**. Như thường lệ, ta cần đọc ngược 4 byte này và cách đọc ngược tương tự như đọc ngược **highCluster** đã trình bày phía trên, con số cuối cùng là kích thước theo đơn vị byte được lưu vào biến **sz**
- Cuối cùng, sau khi thu thập xong thông tin như yêu cầu, ta cập nhật **pByteRoot += 32** để nhảy tới entry kế tiếp cũng như lưu thông tin entry vừa đọc đưa vào mảng kết quả trả về, chính thức kết thúc đọc 1 entry
- Làm tương tự cho các entry kế tiếp thì ta đã đọc xong RDET



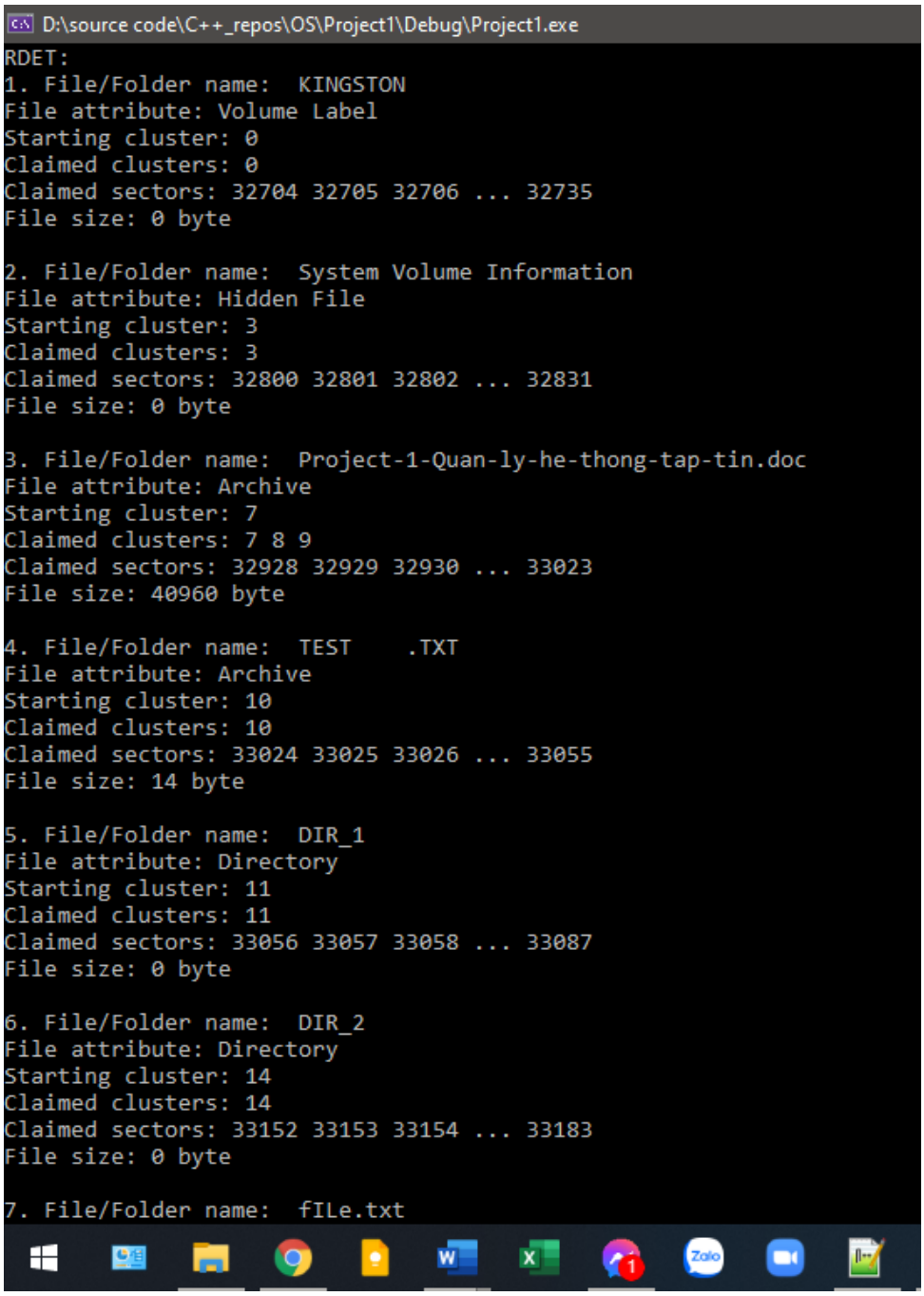
```
                    // chiếm các cluster
                    vector<int> claimedClusters = { startCluster };
                    int cluster = fat[startCluster]; // next cluster index

                    while (cluster < fat.size()) {
                        if (cluster != 0x0 && cluster != 0xffffffff &&
                                              cluster != 0xfffffff &&
                                              cluster != 0xffffff8)
                            claimedClusters.push_back(cluster);
                        else
                            break;
                        cluster = fat[cluster]; //update
                    }
                    f.claimedClusters = claimedClusters;

                    //chiếm các sector
                    vector<int> claimedSectors;
                    int claimedSector = 0;
                    for (int j = 0; j < claimedClusters.size(); ++j) {
                        claimedSector = (Sb + Nf * Sf + Sr) + Sc * (claimedClusters[j] - 2);

                        for(int k=0; k<Sc;++k)
                            claimedSectors.push_back(claimedSector + k);
                    }
                    f.claimedSectors = claimedSectors;

                    // kích thước nội dung tập tin
                    int sz = 0;
                    for (int i = 3; i >= 0; --i)
                        sz = sz << 8 | stRoot.sizeofFile[i];
                    f.fsize = sz;

                    //update pByteRoot
                    pByteRoot += 32;
                    res.push_back(f);
                } //end for
            } //end else: readFile = success
            if (bNoEntry)
                break;
        } while (true);
    } //end if (dwFilePointer != INVALID_SET_FILE_POINTER)
    else
        cout << "INVALID_SET_FILE_POINTER";
}
else
    cout << "cant open handle";
return res;
} //end function
```

c) Hiển thị cây thư mục

- Chức năng: in ra RDET hoặc SDET (với Xdet trong tên hàm mang nghĩa Rdet hoặc Sdet)
- Hàm nhận tham số là 1 mảng các phần tử kiểu **FEntity** và in ra từng phần tử trong mảng với các trường được yêu cầu

```cpp
void printXdet(vector<FEntity> a) {
    for (int i = 0; i < a.size(); ++i) {
        cout << i + 1 << ". File/Folder name:  ";
        for (int j = 0; j < a[i].name.size(); ++j)
            cout << a[i].name[j];

        cout << "\nFile attribute: ";
        if (a[i].attribute & 0x01)
            cout << "Read Only File\n";
        else if (a[i].attribute & 0x02)
            cout << "Hidden File\n";
        else if (a[i].attribute & 0x04)
            cout << "System File\n";
        else if (a[i].attribute & 0x08)
            cout << "Volume Label\n";
        else if (a[i].attribute & 0x10)
            cout << "Directory\n";
        else if (a[i].attribute & 0x20)
            cout << "Archive\n";

        cout << "Starting cluster: " << a[i].startingCluster << endl;

        cout << "Claimed clusters: ";
        for (int j = 0; j < a[i].claimedClusters.size(); ++j)
            cout << a[i].claimedClusters[j] << " ";

        cout << "\nClaimed sectors: ";
        for (int j = 0; j < 3; ++j)
            cout << a[i].claimedSectors[j] << " ";
        cout << "... " << a[i].claimedSectors.back();

        cout << "\nFile size: " << a[i].fsize << " byte";
        cout << "\n\n";
    }
}
```

- Output cây RDET: in lần lượt tên các file/folder có trong ổ đĩa FAT32 theo số thứ tự 1,2,3, … Mỗi file/folder cách nhau bởi 1 hàng. Với mỗi 1 file/folder, các trường sau sẽ được thể hiện:
    - **Số thứ tự**: 1,2,3,...
    - **File/Folder name**: tên file/folder
    - **File attribute**: trạng thái
    - **Starting cluster**: chỉ số cluster bắt đầu
    - **Claimed clusters**: (các) chỉ số cluster chiếm giữ
    - **Claimed sectors**: các chỉ số sector lưu trữ trên đĩa cứng. Vì nếu in ra tất cả sẽ rất dài và lấn nhiều dòng gây thiếu thẩm mỹ nên nhóm quyết định chỉ in 3 chỉ số sector đầu tiên, nối liền sau đó bởi dấu 3 chấm "…" và sector cuối thể hiện là còn các sectors khác. Mỗi sector cách nhau 1 đơn vị
    - **File size**: kích thước tập tin



3. ĐỌC VÀ HIỂN THỊ CÂY THƯ MỤC CON (SDET) CỦA 1 FOLDER BẤT KỲ [Kiệt]

a) Hàm hỗ trợ

- Hàm **convertExtension**: nhận tham số là **extension** dưới dạng mảng BYTE và trả ra **extension** dưới dạng chuỗi string

```
548    string convertExtension(vector<BYTE> extension) {
549        stringstream writer;
550        for (int i = 0; i < extension.size(); i++) {
551            BYTE tmp = extension[i];
552            if (extension[i] > 0x5A) //extension[i] > 'Z'
553                tmp = extension[i] - 0x20;      // In hoa vd: t -> T
554            writer << tmp;
555        }
556        return writer.str();
557    }
558
```

b) Hàm chính

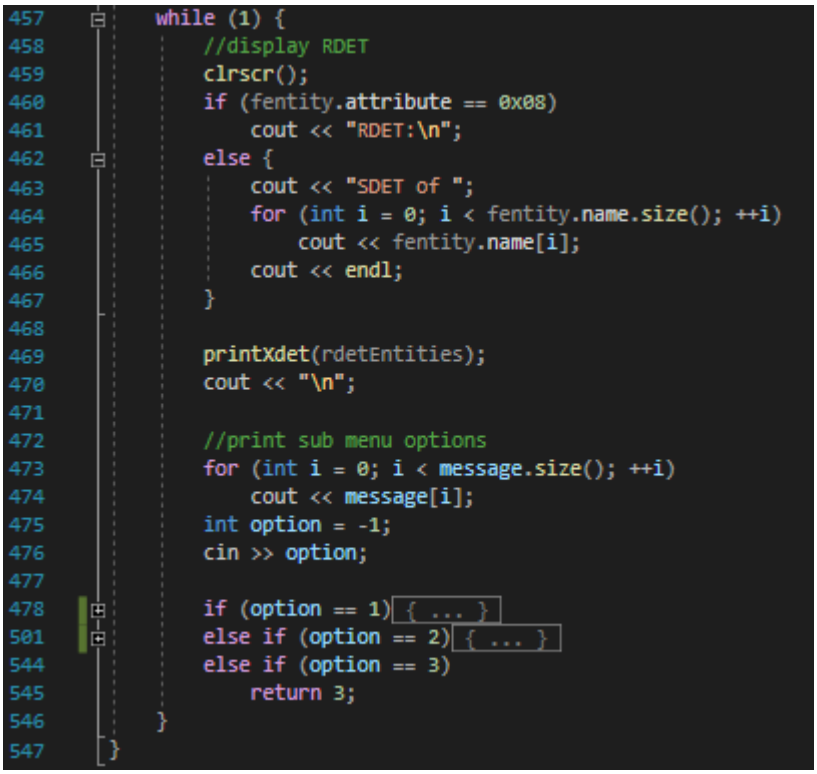
- Hàm **subMenu** tích hợp các chức năng sau: **đọc** và **in** ra cây RDET hoặc SDET của 1 folder **fentity** truyền vào. Như vậy khi gọi hàm chỉ cần gọi 1 lần hàm này thì có thể in được RDET hoặc SDET
- Giải thích hàm và tham số truyền vào"

`int subMenu(vector<FEntity> rdetEntities, BYTE bootSector[512], HANDLE fat32_disk, vector<DWORD> fat, FEntity fentity, vector<int> info)`	
`vector<FEntity> rdetEntities`	Là mảng cha chứa các file/folder
`BYTE bootSector[512]`	Bảng boot sector
`HANDLE fat32_disk`	Handle tới ổ đĩa FAT32 trong máy
`vector<DWORD> fat`	Bảng FAT32
`FEntity fentity`	Directory/folder cần thể hiện SDET
`vector<int> info`	Là thông số cơ bản đọc từ RDET gồm **đúng** các phần tử { Sb, Nf, Sf, Sr, Sc}

- Trả về: lựa chọn 1,2 hay 3 của người dùng nhập vào
- Đầu tiên, khai báo message là 1 tập các chỉ thị dùng để in ra màn hình giúp người dùng đọc và biết các thao tác cần thực hiện. Sb, Nf, Sf, Sr, Sc cũng được khai báo lại nhằm hỗ trợ việc tính vị trí đọc SDET dễ hiểu và dễ dàng hơn

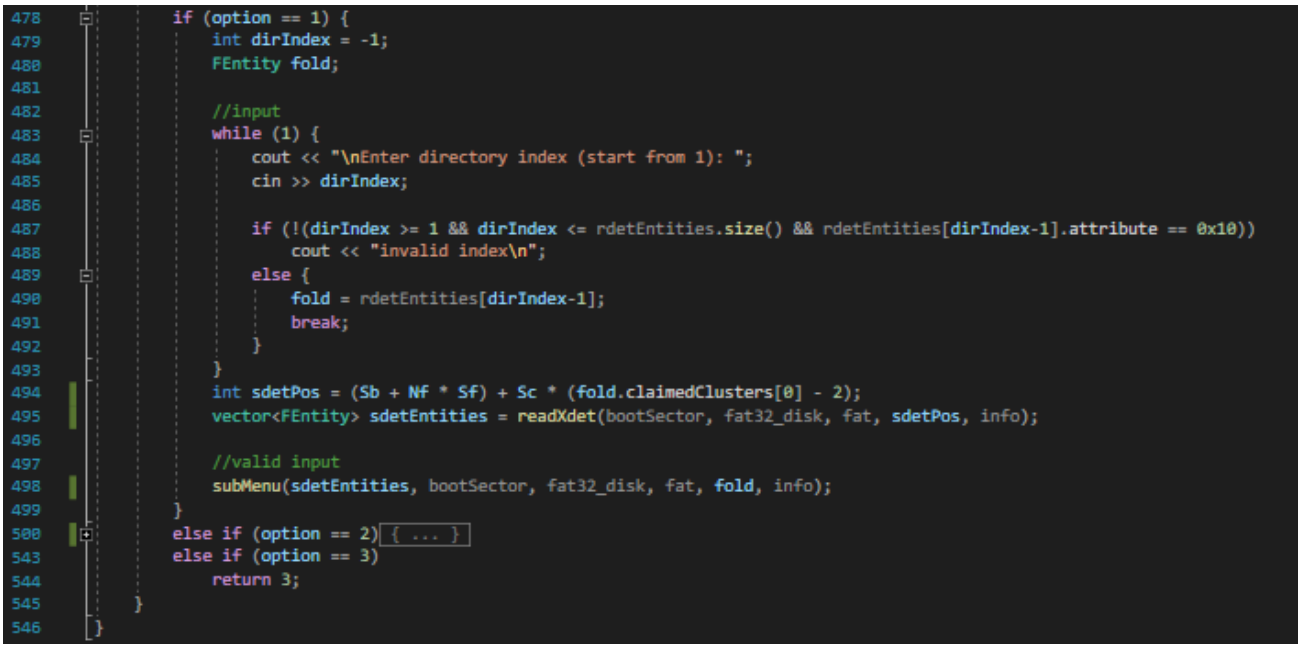
```
449  int subMenu(vector<FEntity> rdetEntities, BYTE bootSector[512], HANDLE fat32_disk, vector<DWORD> fat, FEntity fentity, vector<int> info) {
450      vector<string> message = { "      1. Display SDET\n" ,
451                                 "      2. Display a file's content\n" ,
452                                 "      3. Back\n",
453                                 "      Your choice: " };
454      int Sb = info[0], Nf = info[1], Sf = info[2], Sr = info[3], Sc = info[4];
455      char keyboard = ' ';
456
457      while (1) { ... }
547  }
```

- Trong vòng while, ở dòng 460 - 467, trước khi xử lý đọc và in cây SDET, ta cần in ra cây **RDET** (nếu đang đứng ở Volume ổ đĩa) hoặc **SDET** (nếu đang đứng trong 1 thư mục nào đó) để biết được ta đang ở đâu nếu trường hợp lồng nhiều folder, tránh bị rối khi hiển thị.



```
457     while (1) {
458         //display RDET
459         clrscr();
460         if (fentity.attribute == 0x08)
461             cout << "RDET:\n";
462         else {
463             cout << "SDET of ";
464             for (int i = 0; i < fentity.name.size(); ++i)
465                 cout << fentity.name[i];
466             cout << endl;
467         }
468
469         printXdet(rdetEntities);
470         cout << "\n";
471
472         //print sub menu options
473         for (int i = 0; i < message.size(); ++i)
474             cout << message[i];
475         int option = -1;
476         cin >> option;
477
478         if (option == 1) { ... }
501         else if (option == 2) { ... }
544         else if (option == 3)
545             return 3;
546     }
547 }
```

- Cho người dùng nhập vào 1 trong 3 tùy chọn hiển thị trong **message**. Giả sử nhập 1 (‘**Display SDET**’) → tiếp tục nhập số thứ tự của thư mục muốn người dùng muốn in ra SDET (dòng 484, 485) kết hợp kiểm tra nhập có hợp lệ không (tức chỉ số nhập vào có phải chỉ số của directory hay của cái khác). Sau khi nhập hợp lệ, thư mục **fold** sẽ chính là thư mục có SDET cần đọc
- Dựa vào chỉ số đầu tiên của sector chiếm giữ, tức (Sb + Nf * Sf + 0) + Sc * (K- 2), với K là chỉ số cluster chiếm giữ đầu tiên → đây chính là vị trí SDET **sdetPos** ta cần đi tới để đọc cây thư mục con. Như đã nói ở trên, hàm **readXdet** có thể dùng để đọc thông tin cho cả cây thư mục con với vị trí **sdetPos** được truyền vào. Cách đọc tương tự như đọc RDET đã trình bày ở III.2b
- Sau đó, đệ quy hàm để vào trong thư mục **fold** để in nội dung của cây SDET và làm các thao tác khác. Nếu muốn quay ra ngoài, tức về với thư mục cha chứa thư mục fold, người dùng chỉ cần nhập 3



```
478         if (option == 1) {
479             int dirIndex = -1;
480             FEntity fold;
481
482             //input
483             while (1) {
484                 cout << "\nEnter directory index (start from 1): ";
485                 cin >> dirIndex;
486
487                 if (!(dirIndex >= 1 && dirIndex <= rdetEntities.size() && rdetEntities[dirIndex-1].attribute == 0x10))
488                     cout << "invalid index\n";
489                 else {
490                     fold = rdetEntities[dirIndex-1];
491                     break;
492                 }
493             }
494             int sdetPos = (Sb + Nf * Sf) + Sc * (fold.claimedClusters[0] - 2);
495             vector<FEntity> sdetEntities = readXdet(bootSector, fat32_disk, fat, sdetPos, info);
496
497             //valid input
498             subMenu(sdetEntities, bootSector, fat32_disk, fat, fold, info);
499         }
500         else if (option == 2) { ... }
543         else if (option == 3)
544             return 3;
545     }
546 }
```

- Nếu người dùng nhập 2 (‘**Display a file’s content**’ - in nội dung file txt hoặc báo dùng phần mềm khác nếu file có đuôi khác), yêu cầu người dùng nhập tiếp số thứ tự file mong muốn. Cũng giống như lựa chọn 1, chương trình sẽ kiểm tra input người dùng có hợp lệ ở dòng 509. Nếu hợp lệ, bước tiếp theo là xét xem file có đuôi nào
- Tận dụng thuộc tính **name** của của biến **file** thuộc kiểu FEntity, vòng for được dùng để duyệt tới khi chạm dấu chấm “.” Trong **name** thì các ký tự BYTE khác ký tự 0x00 & 0xff sau dấu chấm sẽ là đuôi của file. Lưu các ký tự đuôi vào mảng **ext**. Kết thúc vòng lặp foro73 dòng 524
- Dòng 525 kiểm tra **ext** có phải là “TXT” hay không nhờ hàm hỗ trợ **convertExtension**, nếu đúng thì đi vào if và đọc nội dung file thông qua hàm **readData** (được trình bày mục 4 ở dưới), sau đó in ra màn hình. Nếu không đúng thì đi vào else ở dòng 523 và in ra thông báo cần dùng phần mềm tương thích để đọc nội dung
- Nếu người dùng nhập 3 (‘**Back**’), chương trình sẽ qua về thư mục cha của thư mục hiện tại đang xét

```
        else if (option == 2) {
            int fileIndex = -1;
            FEntity file;

            //input
            while (1) {
                cout << "\nEnter archive (file) index (start from 1): ";
                cin >> fileIndex;

                if (!(fileIndex >= 1 && fileIndex <= rdetEntities.size() && rdetEntities[fileIndex - 1].attribute == 0x20))
                    cout << "invalid index\n";
                else {
                    file = rdetEntities[fileIndex - 1];
                    //test
                    //tim đuôi
                    vector<BYTE> ext;
                    for (int i = 0; i < file.name.size(); ++i) {
                        if (file.name[i] == '.') {
                            for(int j=i+1; j< file.name.size(); j++)
                                if(file.name[j] != 0x00 && file.name[j] != 0xff)
                                    ext.push_back(file.name[j]);
                            break;
                        }
                    }

                    if (convertExtension(ext) == "TXT") {
                        int dataPos = Sb + Nf * Sf + Sc * (file.claimedClusters[0] - 2);
                        readData(fat32_disk, Sc, dataPos, file);

                        cout << "File's data:\n\t";
                        for (int i = 0; i < file.data.size(); ++i)
                            cout << file.data[i];
                    }
                    else
                        cout << "Use other apps to open this file.";

                    cout << "\nPress any key to escape";
                    keyboard = _getch();
                    //cin >> keyboard;
                    break;
                }
            }
        }
        else if (option == 3)
            return 3;
    }
}
```

4. ĐỌC VÀ HIỂN THỊ NỘI DUNG TẬP TIN

a) Đọc nội dung tập tin

- Như có đề cập ở phần 3 hàm **readData** sẽ được sử dụng để đọc dữ liệu của một tập tin có phần mở rộng *.txt. Hàm được mô tả như sau:

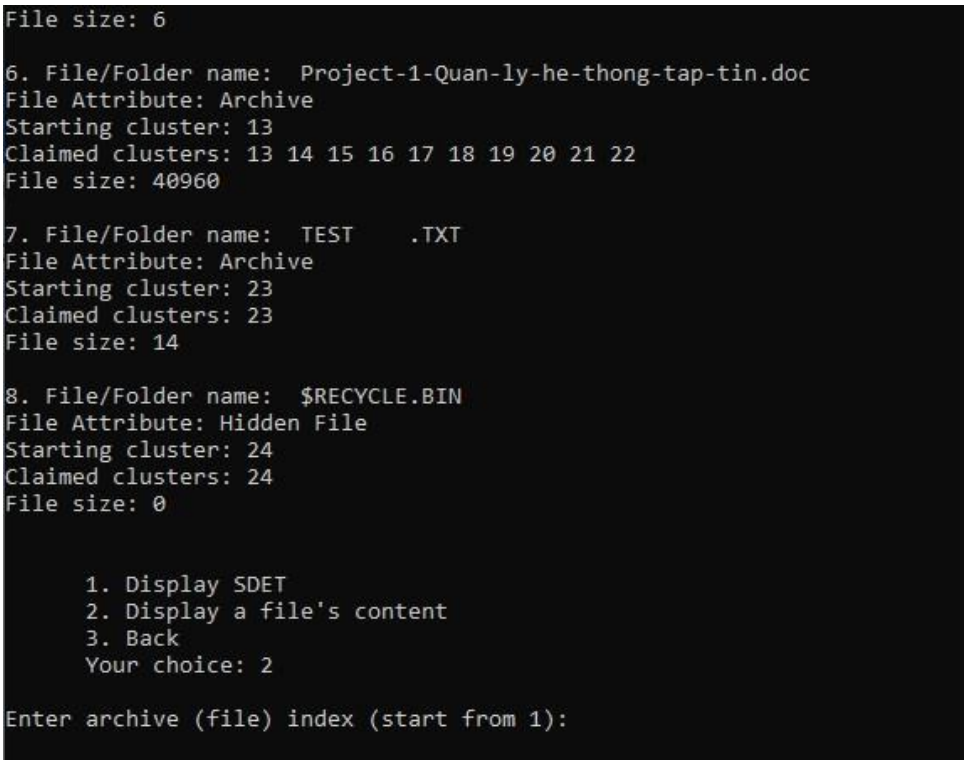
`void readData(HANDLE fat32_disk, int Sc, int readPos, FEntity& file)`	
`HANDLE fat32_disk`	Handle ổ đĩa FAT32 trong máy
`int Sc`	Số Sector/Cluster
`int readPos`	Vị trí bắt đầu sector chứa dữ liệu của tập tin, được tính bằng (Sb + Nf * Sf + 0) + Sc * (K- 2)
`FEntity& file`	Tập tin *.txt cần đọc nội dung

```
159  void readData(HANDLE fat32_disk, int Sc, int readPos, FEntity& file) {
160      BYTE byteRoot[512];
161      DWORD dwBytesRead;
162      DWORD dwFilePointer = SetFilePointer(fat32_disk, (512 * readPos), NULL, FILE_BEGIN);
163      int nSector = file.claimedClusters.size() * Sc;
164
165      if (dwFilePointer != INVALID_SET_FILE_POINTER) {
166          BOOL noMoreData = FALSE;
167          int i = 0;
168          for (i = 0; i < nSector; i++) {
169              if (!ReadFile(fat32_disk, byteRoot, 512, &dwBytesRead, NULL))
170                  cout << "Error when reading Data.\n";
171
172              // check entry có trống
173              if (byteRoot[0] == 0x00) {
174                  return;
175              }
176
177              for (int i = 0; i < 512; i++) {
178                  if (byteRoot[i] == 0x00) {
179                      noMoreData = true;
180                      break;
181                  }
182                  file.data.push_back(byteRoot[i]);
183              }
184              if (noMoreData) break;
185          }
186      }
187      else
188          cout << "INVALID_SET_FILE_POINTER";
189  }
```

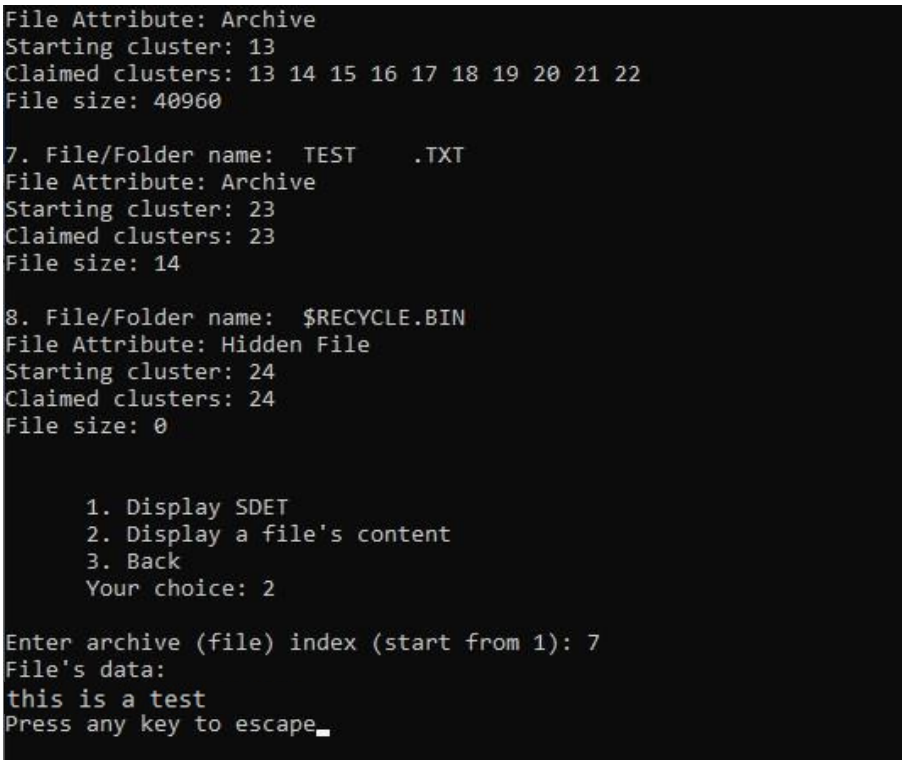
- **byteRoot** là vùng chứa dữ liệu tạm thời được đọc từ vùng DATA. Kích thước 512 bytes tương ứng với 1 sector.
- **dwBytesRead** sẽ ghi nhận số byte được đọc. Hàm SetFilePointer sẽ di chuyển con trỏ đọc file đến vị trí readPos để đọc vùng dữ liệu. Vị trí bytes bắt đầu vùng dữ liệu của tập tin sẽ được lưu vào dwFilePointer.
- **nSector** tính số lượng sector mà tập tin chiếm bằng cách nhân *số lượng cluster tập tin chiếm* thông qua đọc bảng FAT với *số sector trên cluster (hay **Sc**)*.
- Nếu **dwFilePointer** hợp lệ các câu lệnh bên trong sẽ được thực thi. Vòng lặp for sẽ quét qua toàn bộ sector mà tập tin này chiếm. Tiến hành đọc từng sector và lưu trữ tạm thời vào **byteRoot**. Nếu việc đọc file thất bại sẽ xuất ra màn hình “Error when reading Data.”;
- Nếu đọc file thành công, thuộc tính data của file sẽ tiến hành lưu từng byte một. Nếu gặp ký tự NULL (0x00) thì quá trình đọc nội dung kết thúc. Cờ **noMoreData** sẽ được bật để thoát khỏi vòng lặp. Kết thúc hàm, hoàn tất quá trình đọc dữ liệu.

b) Hiển thị nội dung tập tin

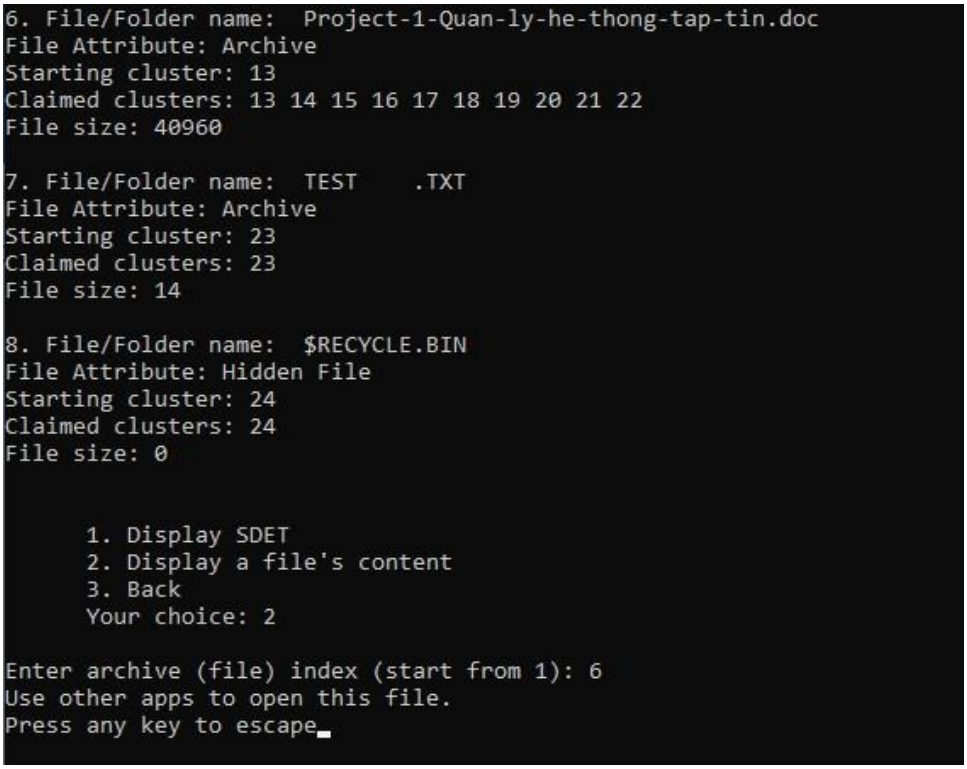
- Đây là lựa chọn 2 ở hàm **subMenu**. Khi lựa chọn 2 được chọn hệ thống sẽ yêu cầu nhập số thứ tự của tập tin muốn hiển thị thông tin



- Ví dụ muốn coi nội dung của tập tin **TEST.TXT** → nhập 7. Sau khi chọn xong chỉ số tập tin, trong trường hợp tập tin được chọn có phần mở rộng *.txt thì nội dung tập tin sẽ được đọc lên và hiển thị ra màn hình bắt đầu từ dưới dòng "**File's data**". Ở đây, nội dung file **TEST.TXT** là "this is a test"



- Nếu tập tin được chọn có phần mở rộng khác *.txt thì sẽ in ra thông báo sử dụng phần mềm tương thích để đọc nội dung.



IV. SO SÁNH SỰ KHÁC NHAU GIỮA NTFS VÀ FAT32

- FAT32 sử dụng không gian địa chỉ 32 bit còn NTFS sử dụng không gian địa chỉ 64 bit
- FAT32 đơn giản trong khi cấu trúc NTFS khá phức tạp.
- NTFS có thể hỗ trợ kích thước tệp và âm lượng lớn hơn cùng với tên tệp lớn so với hệ thống tệp FAT32.
- FAT32 không cung cấp mã hóa và bảo mật nhiều trong khi NTFS được kích hoạt với bảo mật và mã hóa.
- Khá dễ dàng để chuyển đổi một hệ thống tệp FAT thành một hệ thống khác mà không mất dữ liệu. Ngược lại, chuyển đổi NTFS rất khó đạt được.
- Hiệu năng NTFS tương đối tốt hơn so với FAT32 vì nó cũng cung cấp khả năng chịu lỗi.
- Các tập tin được truy cập nhanh hơn trong trường hợp NTFS. Ngược lại, FAT32 chậm hơn NTFS.
- NTFS truyền các tính năng như ghi nhật ký và nén, không được cung cấp bởi FAT32

Cơ sở để so sánh	FAT32	NTFS
Căn bản	Cấu trúc đơn giản	Cấu trúc phức tạp

Số lượng ký tự tối đa được hỗ trợ trong một tên tệp	83	255
Kích thước tệp tối đa	4GB	16TB
Mã hóa	Không cung cấp	Cung cấp
Bảo vệ	Dạng kết nối	Địa phương và mạng
Chuyển đổi	Được phép	Không cho phép
Chịu lỗi	Không có quy định cho khả năng chịu lỗi.	Tự động khắc phục sự cố
Khả năng tương thích với các hệ điều hành	Phiên bản windows cũ- Win 95/98 / 2K / 2K3 / XP	Các phiên bản mới hơn - Giành NT / 2K / XP / Vista / 7
Danh sách điều khiển truy cập	Không	Vâng
Dung lượng đĩa người dùng	Không	Vâng
Nhật ký và nhật ký kênh	Vắng mặt	Cung cấp nhật ký để theo dõi các hoạt động trước đó.
Hiệu suất	Tốt	Tốt hơn so với FAT32
Liên kết cứng và mềm	Không có mặt	Chứa đựng
Tốc độ truy cập	Ít tương đối	Hơn
Nén	Không cung cấp nén.	Hỗ trợ nén file.

V. NGUỒN THAM KHẢO

- Đọc bootsector NTFS: http://ntfs.com/ntfs-partition-boot-sector.htm
- http://diendan.congdongcviet.com/archive/index.php/t-5058.html
- https://123docz.net/document/21079-thuc-hanh-he-dieu-hanh-3-floppydisk-doc.htm
- Đọc bảng FAT: https://stackoverflow.com/questions/45833495/how-to-read-fat-table-in-c

VI. LINK VIDEO DEMO

https://youtu.be/1-9YwBhyUXQ