

# Báo cáo đồ án 21

## I . Thành viên:

MSSV	Họ và tên
18127121	Nguyễn Đăng Khoa
18127134	Lê Huỳnh Long

## II. Nội dung báo cáo:

### Nội dung đồ án:

Xây dựng một mô hình tập tin và thiết kế kiến trúc tổ chức cho một volume file trong 1 file (volume file đó tương tự như 1 file .ISO hoặc .ZIP). VCT thực hiện các chức năng: tạo vol, xem danh sách file trong vol, đặt mật khẩu truy xuất cho 1 file trong vol, chép 1 file trong vol ra ngoài (export), chép 1 file từ bên ngoài vào vol (import), xóa 1 file trong vol.

### Cấu trúc của của một item (file/folder):

```
struct File
{
    string name;        // Tên file
    string extension;    // Đuôi mở rộng
    int firstCluster;    // cluster đầu
    long size;           // Kích thước file, nếu là folder thì là số entry
    string password = ""; // Password của file
    bool att;            // Thuộc tính, nếu att == 1 thì là folder, ngược lại là file
    bool isPassword;     // Kiểm tra có chứa password hay ko
};
```

- Name: tên của item
- Extension: đuôi của item
- firstCluster: là cluster bắt đầu của item, sau đó truy cập bảng FAT để lấy các cluster item đó chiếm.
- size: kích thước của item, nếu là folder thì đó là số entry của folder.
- password: là mật khẩu do người dùng đặt (độ dài từ 0 đến 12).
- att: thuộc tính của item, nếu là file thì gán bằng 0, nếu là folder thì gán bằng 1
- isPassword: biến kiểm tra có chứa password hay không, nếu có thì bằng 1, nếu không thì bằng 0.

## Cấu trúc của một entry

Entry sẽ hiển thị thông tin của item. Một entry gồm entry chính và entry phụ, mỗi entry sẽ có 32 bytes.

Entry chính sẽ là một cấu trúc File (đã trình bày ở trên) gồm tên (tên ngắn gồm 8 ký tự), đuôi mở rộng (3 ký tự), ...

Entry phụ dùng để lưu tên đầy đủ, không bị cắt. Dấu hiệu nhận biết entry phụ qua “.” (dấu chấm và một khoảng cách).

Ví dụ: một file có tên abcdefgh1234.abc thì entry chính lưu tên ngắn là “abcdefgh”, entry phụ sẽ lưu là “abcdefgh1234.abc”

Ví dụ với HxD:

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000A00	BE	20	74	65	73	74	76	6F	6C	75	6D	65	20	20	20	20	[ ] testvolume                      Sector 5
00000A10	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
00000A20	74	65	73	74	76	6F	6C	75	20	20	20	31	30	00	FE	FE	testvolu    10.đđ
00000A30	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	03	00	00	01	00	00	đđđđđđđđđđ.....

## Tất cả hàm trong các file header:

### BootSector.h:

Là nơi lưu các thông tin quan trọng của file như volume size, số sector của một volume, size của sector, số entry của bảng RDET, size của bảng FAT, số sector trong một cluster, size của cluster. Vùng bootsector chỉ chiếm 1 sector (sector đầu).

```
int volSize; // Volume size: kích thước của volume, đơn vị MB
int volSector; // Số sector của volume
int sectorSize; // Size của mỗi sector, mặc định sẽ là 512 byte
int clusterSize; // mặc định sẽ là 4096 (dạng  $2^{(10 + n)}$ )
int clusterSectors; // Sc: Số sector của mỗi cluster, clusterSize / sectorSize
int bootSize; // Sb, mặc định = 1 sector
int fatSize; // SF: số sector của bảng FAT, dựa trên số cluster hiện có
int entrySizeRDET; // SR: số entry của RDET, mặc định là 512 entry (Mỗi entry có 32 bytes)
int currentVolSector; // Số sector còn lại của volume;
```

## FAT.h:

Bảng quản lý cluster của các item. Có nhiệm vụ truy xuất cluster của một item.

Kích thước bảng FAT tính bằng công thức  $fatLen \leq \lceil \frac{volSec - SB + SR}{256 * Sc + 1} \rceil$

```
FAT(BootSector& bs) { ... }  
int getCluster(int k) { ... }  
vector<int> findEmptyOffsets(fstream& f, WIN32_FIND_DATA file, long sizeFize) { ... }  
void writeFAT(fstream& f, vector<int> clusters) { ... }  
vector<int> getItemClusters(fstream& f, int clusterK) { ... }  
void deleteItem(fstream& f, int clusterK) { ... }
```

```
BootSector() { ... }  
BootSector(int volSize) { ... }  
void createBootSector(fstream& f) { ... }  
void readBootSector(fstream& f) { ... }  
int getVolumeSize() { ... }  
int getClusterSector() { ... }  
int getRDETOffset() { ... }  
int getRDETSIZE() { ... }  
int getFATOffset() { ... }  
int getSectorSize() { ... }  
int getFATSize() { ... }  
int getCurrentSize() { ... }  
void printBootSector() { ... }
```

Ví dụ thực tế bằng HxD, sector đầu trong bảng FAT

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text	
00000200	10	00	FF	00	FF	00	08	00	0A	00	0C	00	0E	00	FF	00	ÿ.ÿ.ÿ.....ÿ.	Sector 1
00000210	12	00	14	00	16	00	18	00	1A	00	1C	00	1E	00	20	00	.....	
00000220	22	00	24	00	26	00	28	00	2A	00	2C	00	2E	00	30	00	".\$.&.(.*.,...0.	
00000230	32	00	34	00	36	00	38	00	3A	00	3C	00	3E	00	40	00	2.4.6.8.:.<.>.@.	
00000240	42	00	44	00	46	00	48	00	4A	00	4C	00	4E	00	50	00	B.D.F.H.J.L.N.P.	
00000250	52	00	54	00	56	00	58	00	5A	00	5C	00	5E	00	60	00	R.T.V.X.Z.\.^.`.	
00000260	62	00	64	00	66	00	68	00	6A	00	6C	00	6E	00	70	00	b.d.f.h.j.l.n.p.	
00000270	72	00	74	00	76	00	78	00	7A	00	7C	00	7E	00	80	00	r.t.v.x.z. .~.€.	
00000280	82	00	84	00	86	00	88	00	8A	00	8C	00	8E	00	90	00	,...t."...E.....	
00000290	92	00	94	00	96	00	98	00	9A	00	9C	00	9E	00	A0	00	'."-."...æ....	
000002A0	A2	00	A4	00	A6	00	A8	00	AA	00	AC	00	AE	00	B0	00	o.µ.¡.¡."~.~.ø.°.	
000002B0	B2	00	B4	00	B6	00	B8	00	BA	00	BC	00	BE	00	C0	00	°.°q.°..°µ.°À.	
000002C0	C2	00	C4	00	C6	00	C8	00	CA	00	CC	00	CE	00	D0	00	À.À.Æ.È.Ê. Ì.Î.Ð.	
000002D0	D2	00	D4	00	D6	00	D8	00	DA	00	DC	00	DE	00	E0	00	'.'Ö.Ö.Ø.Ú.Û. Ò.à.	
000002E0	E2	00	E4	00	E6	00	E8	00	EA	00	EC	00	EE	00	F0	00	ä.ä.æ.è.ê. ì.î.ð.	
000002F0	F2	00	F4	00	F6	00	F8	00	FA	00	FC	00	FE	00	00	01	..ö.ö.ø.ú.ü.ä...	
00000300	02	01	04	01	06	01	08	01	0A	01	0C	01	0E	01	10	01	.....	
00000310	12	01	14	01	16	01	18	01	1A	01	1C	01	1E	01	20	01	.....	
00000320	FF	00	FF	00	FF	00	28	01	2A	01	2C	01	2E	01	30	01	ÿ.ÿ.ÿ.(.*.,...0.	
00000330	32	01	34	01	36	01	38	01	3A	01	3C	01	3E	01	40	01	2.4.6.8.:.<.>.@.	
00000340	42	01	44	01	46	01	48	01	4A	01	4C	01	4E	01	50	01	B.D.F.H.J.L.N.P.	
00000350	52	01	54	01	56	01	58	01	5A	01	5C	01	5E	01	60	01	R.T.V.X.Z.\.^.`.	
00000360	FF	00	FF	00	FF	00	00	00	00	00	00	00	00	00	00	00	ÿ.ÿ.ÿ.....	
00000370	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....	
00000380	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....	
00000390	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....	
000003A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....	
000003B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....	
000003C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....	
000003D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....	
000003E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....	
000003F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....	

## RDET.h:

Quản lý các item, truy xuất folder/file để ghi vào volume, đọc ghi entry, truy xuất item trong một folder, ghi file, chép file ra ngoài volume, xóa một item hoàn toàn.

Kích thước bảng RDET mặc định sẽ là 512 entry (mỗi entry 32 bytes)

⇒ RDET size = 512 \* 32 / (sector size)

```

int getTotalEmptyCluster() { ... }
string handleItemName(File n) { ... }
string readItemInfo(WIN32_FIND_DATA file, string password) { ... }
string getSubEntry(string fileName) { ... }
int getSizeOfFolder(string path) { ... }
void getAllItemsWithinFolder(fstream& f, string folder, int pivotOffset, FAT& fat, string password) { ... }
long getTotalSize(string path) { ... }
void addItem(fstream& f, string item, int firstCluster, bool isFolder, FAT& fat, bool isPassword) { ... }
int getCluster(int k) { ... }
long getFileSize(WIN32_FIND_DATA fd) { ... }
void exportFile(fstream& f, fstream& out, FAT& fat, int clusterK, long fileSize) { ... }
void exportFolder(fstream& f, string path, FAT& fat, File folder) { ... }
void deleteItemContent(fstream& f, FAT& fat, int clusterK) { ... }
void deleteItem(fstream& f, FAT& fat, int clusterK, int folderCluster) { ... }
vector<File> getSubItems(fstream& f, FAT& fat, int clusterK = 0) { ... }
int showFolder(fstream& f, FAT& fat, int clusterK = 0) { ... }

```

```

void addFile(fstream& f, WIN32_FIND_DATA file, int pivotCluster, FAT& fat, string path, string password){ ... }
void addFolder(fstream& f, string folder, int startCluster, FAT& fat, string password){ ... }
string getFileExtension(string fileName){ ... }
string getShortName(WIN32_FIND_DATA fileName){ ... }
WIN32_FIND_DATA processShortName(WIN32_FIND_DATA fileName){ ... }
void writeFileContent(fstream& volume, vector<int> clustersOffset, string path, long fileSize){ ... }

```

## FileManagement.h:

Đây chính là file xử lý giao diện cho chương trình, hiển thị menu, các lựa chọn, chức năng của chương trình.

```

FileManagement(long volSize, string path){ ... }
FileManagement(string path){ ... }
~FileManagement(){ ... }
bool checkFileExists(string dirName){ ... }
bool checkDirectoryExists(string dirName){ ... }
void addItem(int containingFolderCluster, int clusterK){ ... }
void showFolder(vector<File> allItems){ ... }
void exportItem(File item, string path_out){ ... }
void deleteItem(File item, int containingFolderCluster){ ... }

void showMenu(){ ... }

```

## Các hàm quan trọng:

### RDET.h: addFile và addFolder

```

void addFile(fstream& f, WIN32_FIND_DATA file, int pivotCluster, FAT& fat, string path, string password){ ... }
void addFolder(fstream& f, string folder, int startCluster, FAT& fat, string password){ ... }

```

Hai hàm này rất quan trọng trong việc hỗ trợ chức năng import. Có thể có password hoặc không có. Cả hai hàm đều nhờ đến bảng FAT để ghi các cluster nó chiếm.

addFolder sẽ import một folder bên ngoài vào folder mà người dùng đang truy cập. Hàm này sẽ đệ quy để đi tìm các sub Folder và sub File bên trong nó, sub file sẽ được import thông qua hàm addFile, khi tìm được một file hay folder con nào sẽ tiến hành tìm entry trống để ghi thông tin file/folder đó, sau đó tiếp tục vét cạn để ghi đến khi hết thì thôi.

addFile có nhiệm vụ thêm một file vào volume, ghi nội dung của file vào các cluster được lấy từ bảng FAT thông qua cluster bắt đầu của file đó là tham số pivotCluster.

Có cơ chế kiểm tra xem liệu volume còn đủ dung lượng để import vào không.

## RDET.h: exportFile và exportFolder

```
void exportFile(fstream& f, fstream& out, FAT& fat, int clusterK, long fileSize) { ... }  
void exportFolder(fstream& f, string path, FAT& fat, File folder) { ... }
```

Tương tự như hai hàm import. Hàm export sẽ chép một file bên trong volume ra bên ngoài, theo đường dẫn mà người sử dụng nhập. Nếu là folder thì sẽ có cả những file và folder con trong đó. Sau khi chép item ra ngoài thì item vẫn còn trong volume mà không bị xóa (tương tự copy item ra ngoài).

Hàm này sẽ truy vấn tới bảng FAT để tìm kiếm các cluster của file/folder chiếm thông qua cluster bắt đầu cluster. Khi có thông tin các cluster mà file/folder đó chiếm thì sẽ truy xuất đến vị trí các cluster đó và chép nội dung file ra ngoài thông qua tham số fstream& out.

## RDET.h: deleteItemContent và deleteItem

```
void deleteItemContent(fstream& f, FAT& fat, int clusterK) { ... }  
void deleteItem(fstream& f, FAT& fat, int clusterK, int folderCluster) { ... }
```

Hai hàm này sẽ xóa item mất hẳn ra khỏi volume (xóa hoàn toàn).

Hàm deleteItem sẽ xóa thông tin entry của file/folder đó, nếu là folder thì sẽ đệ quy để tìm sub files/folder. Sau đó dùng hàm deleteItemContent để xóa các nội dung của file thông qua các cluster của file đó như các hàm nêu ở trên (dùng FAT).

Hàm deleteItemContent cũng gọi fat để lấy các vị trí của item đó bằng các cluster item đó chiếm, sau đó nhảy tới offset của nó để ghi đè '\0' vào dữ liệu của item.

## RDET.h: getAllItemsWithinFolders

```
void getAllItemsWithinFolder(fstream& f, string folder, int pivotOffset, FAT& fat, string password)
```

Hàm này có chức năng tìm kiếm mọi items trong một folder, bên trong hàm này sẽ đệ quy khi bắt gặp một sub folder để có thể tìm tiếp mọi items trong sub folder đó. Sau đó các item sẽ được add vào volume bằng hàm addFile hoặc addFolder.

Các item có kiểu dữ liệu là WIN32\_FIND\_DATA của win32 api.

## RDET.h: readItemInfo

```
string readItemInfo(WIN32_FIND_DATA file, string password)
```

Hàm này có chức năng đọc thông tin từ một WIN32\_FIND\_DATA file để trả về một string có 26 ký tự tương thích với thông tin của file đó như tên, đuôi mở rộng, password, thuộc tính file hay folder. Sau đó chuỗi này sẽ được lưu vào entry chính của RDET.

## RDET.h: getSubEntry

```
string getSubEntry(string fileName)
```

Hàm này có chức năng lấy tên dài của item để biến thành một entry phụ thể hiện tên dài của item đó.

## FAT.h: findEmptyOffsets

```
vector<int> findEmptyOffsets(fstream& f, WIN32_FIND_DATA file, long sizeFize)
```

Hàm tìm các cluster trống cho một item. Trả về một mảng các cluster trống của item đó.

## FAT.h: writeFat

```
void writeFAT(fstream& f, vector<int> clusters)
```

Hàm ghi các cluster trống vào bảng FAT. Kết thúc cluster của item bằng FF (255).

## FileManagement.h: showMenu

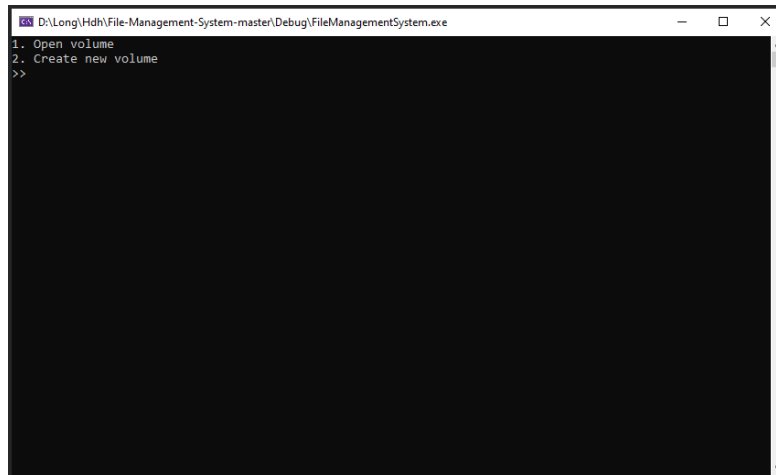
```
void showMenu()
```

Hàm này dùng để thể hiện tất cả các chức năng của chương trình, là giao diện chính của chương trình.

### III. Cách chạy chương trình:

Mở chương trình bằng Visual Studio và chạy chương trình.

Giao diện console của chương trình:



```
D:\Long\Hdh\File-Management-System-master\Debug\FileManagementSystem.exe
1. Open volume
2. Create new volume
>>
```

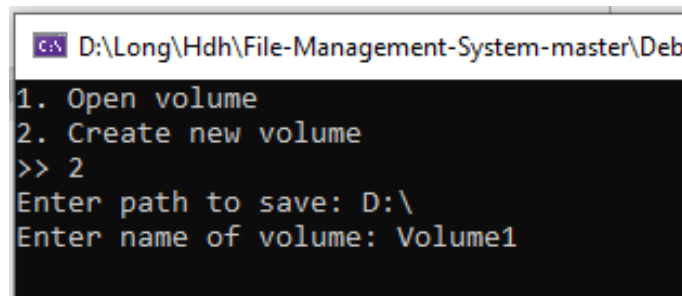
*Có 2 cách để mở volume:*

1. Chọn 1 volume có sẵn: Ấn 1 và Enter. File volume được chọn phải có đuôi là .re



```
1. Open volume
2. Create new volume
>> 1
Link of volume (*.re file): D:/test.re_
```

2. Tạo 1 volume mới: Ấn 2 và Enter. Nếu tạo mới, chương trình sẽ yêu cầu nhập đường dẫn để Save volume và nhập tên volume. Sau đó, nhập kích thước volume cần tạo (mặc định **MB**). Kích thước không giới hạn, kích thước càng lớn tạo càng lâu. Tên volume sẽ được gán thêm đuôi “.re”.



```
D:\Long\Hdh\File-Management-System-master\Deb
1. Open volume
2. Create new volume
>> 2
Enter path to save: D:\
Enter name of volume: Volume1
```



```
C:\> D:\Long\Hdh\File-Management-System-master\Debug\FileMan:
Enter size of volume(MB): 100
```

**Nếu đường dẫn sai sẽ thông báo và cho người dùng nhập lại**

Sau khi chọn mở hoặc tạo file, chương trình sẽ hiển thị 5 lựa chọn:

```
-----
1. Truy cập folder
2. Import file/folder vào Folder gốc
3. Export file/folder
4. Xóa item
5. Quay về folder trước
>>
```

1. Truy cập folder con nằm trong folder đang truy cập.
2. Import file hoặc folder vào folder đang truy cập.
3. Export file hoặc folder ra khỏi volume.
4. Xóa 1 file hoặc folder ra khỏi volume.
5. Nếu truy cập vào folder con, sử dụng chức năng này để quay lại folder cũ.  
Volume chưa có gì nên không thể dùng lệnh truy cập folder.

Ta **Import** một folder có sẵn vào volume (BackGround) bằng cách nhấn phím 2. Chọn File hoặc Folder để nhập (trong trường hợp này là Folder) và nhập đường dẫn. Sau đó chọn có đặt password cho item vừa import không.

Nếu file/folder đó có password, để thao tác với item đó thì phải nhập password, sau đó chương trình sẽ kiểm tra, nếu đúng thì mới cho truy cập.

```
C:\> D:\Long\Hdh\File-Management-System-master\Debug\FileManagementSystem.exe
Enter size of volume(MB): 700
Dang truy cap: Folder goc
Name
-----
1. Truy cap folder
2. Import file/folder vao Folder goc
3. Export file/folder
4. Xoa item
5. Quay ve folder truoc
>> 2
IMPORT ITEM
Chon import file hay folder (1: file, 2: folder): 2
Chon duong dan: D:\BackGround
Co dat password khong?(1: Co, 0: Khong): 0
```

Sau khi **Import** sẽ cho kết quả sau:

```
D:\Long\Hdh\File-Management-System-master\Debug\FileManagementSystem.exe
Dang truy cap: Folder goc
Name                                     Type      Size (Bytes)
1. BackGround                           Folder
-----
1. Truy cap folder
2. Import file/folder vao Folder goc
3. Export file/folder
4. Xoa item
5. Quay ve folder truoc
>>
```

Sau đó, **ấn 1 để Truy cập folder**, chọn số thứ tự đứng trước folder muốn truy cập và nhập. Ở đây là 1 (BackGround). Sau khi truy cập sẽ hiện ra list các file trong folder đó.

```
D:\Long\Hdh\File-Management-System-master\Debug\FileManagementSystem.exe
Dang truy cap: Background

Name                                     Type      Size (Bytes)
1. 18925_en_1.jfif                      jfi       1209846
2. 18930_en_1.jfif                      jfi       1312271
3. 18931_en_1.jfif                      jfi       802632
4. 18953_en_1.jfif                      jfi       2893184
5. 18959_en_1.jfif                      jfi       769157
6. 18966_en_1.jfif                      jfi       2909210
7. 19049_en_1.jfif                      jfi       1133070
8. 19070_en_1.jfif                      jfi       1057219
9. 19098_en_1.jfif                      jfi       630873
10. 19119_en_1.jfif                     jfi       2413161
11. 19127_en_1.jfif                     jfi       777216
12. 19161_en_1.jfif                     jfi       802244

-----
1. Truy cap folder
2. Import file/folder vao Background
3. Export file/folder
4. Xoa item
5. Quay ve folder truoc
>> 3
Chon item muon export:
```

Ở đây có thể chọn **Export** hoặc **Xóa**. Tương tự như **truy cập folder**, chương trình sẽ cho ta nhập số thứ tự của item muốn **Export** hoặc **Xóa**. Khi **Export** ra bên ngoài thì item vẫn còn đó và chương trình sẽ yêu cầu 1 đường dẫn để Export file đến đó. Còn khi **Xóa** thì file sẽ mất hẳn.

Sau khi thao tác xong, ta có thể chọn phím 5 để quay lại folder trước hoặc tiếp tục chọn chức năng khác.

## IV. Đánh giá chương trình

### Ưu điểm:

- Chương trình có thể tạo Volume không giới hạn.
- Chương trình có thể kiểm tra đường dẫn file hoặc folder nhập vào có lỗi hay không.
- Chương trình có thể mở và đọc volume một cách chính xác.
- Chương trình có thể Import và Export các file hoặc thư mục từ trong Vol ra ngoài và ngược lại.
- Chương trình có thể xóa item một cách hoàn toàn (bảng FAT trở về 0 và nội dung của item đó thành 0).

⇒ **Chương trình đáp ứng được yêu cầu đề bài.**

### Hạn chế:

- Bug nhỏ khi import một item có kích thước lớn (> 100 MB).
- Chưa hash password.
- Tạo volume với kích thước lớn khá lâu (tồn vài giây).
- Giao diện đơn sơ.

### V. Bảng phân công công việc

Tên	Công Việc	Mức độ hoàn thành
Nguyễn Đăng Khoa	FileManagement.h RDET.h	50%
Lê Huỳnh Long	BootSector.h FAT.h	50%
<b>Tổng mức độ hoàn thành</b>		<b>100%</b>