TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
ĐẠI HỌC QUỐC GIA TP.HCM

KHOA CÔNG NGHỆ THÔNG TIN

ĐỒ ÁN GIỮA KÌ – HỆ THỐNG TẬP TIN

HỆ ĐIỀU HÀNH

Lớp: 21CLC03

Sinh viên: Trần Nguyên Huân – 21127050

Nguyễn Hoàng Phúc - 21127671

Giảng viên hướng dẫn: Thái Hùng Văn, Đặng Trần Minh Hậu

THÀNH PHỐ HỒ CHÍ MINH - 2023

Mục Lục

I. Bảng phân công công việc

MSSV	Họ và tên	Công việc	Tỉ lệ công việc
21127050	Trần Nguyên Huân	- Code câu 1 - Viết báo cáo câu 1, câu 2A	50%
21127671	Nguyễn Hoàng Phúc	- Code câu 2 - Viết báo cáo câu 2B	50%

*** Chú ý:**

- Code câu 1 và câu 2 có thể chạy được trên Visual Studio, các trình biên dịch trên IDE khác thì có thể không chạy được.

II. Câu 1

II.1. Câu 1 – A: Viết chương trình tạo vào 100 tập tin tự động trên thư mục gốc mỗi volume

II.1.1. Hàm tạo 1 file .DAT

```
void create_DAT_file(string file_name, int cluster_size, int byte_per_sector, int i) {
    int digits = digitsOfNumber(2020 + i);
    int cluster = 4 - i % 4;
    int byte = cluster * cluster_size * byte_per_sector; // tinh kich thuoc byte tuong ung voi so cluster
    fstream f;
    f.open(file_name, ios::out);
    if (cluster == 1) { //neu cluster = 1 thi dien vo 1 dong
        f << 2020 + i << endl;
    }
    for (int j = 0; j < (byte / (digits + 2)) / 2 + 1; j++) { //giai thich trong bao cao
        f << 2020 + i << endl;
    }
    f.close();
}
```

- Tham số truyền vào:

+ Tên file (file_name): dưới dạng string lưu đường dẫn của tập tin. VD: E:\F0.DAT

+ Chỉ số i dưới dạng kiểu số nguyên tương ứng với file Fi thì sẽ ghi vào file các dòng 2020 + i

- Chú thích 2 hàm tiện ích để lấy thông tin volume (thuộc thư viện Windows.h)

```
TCHAR volume[MAX_PATH + 1] = { 0 };
DWORD sectorsPerCluster = 0;
DWORD bytesPerSector = 0;
DWORD reservedSectors = 0;
DWORD numberOfFATs = 0;
ULARGE_INTEGER totalBytes = { 0 };
ULARGE_INTEGER freeBytes = { 0 };
DWORD sectorsPerFAT = 0;
DWORD rootDirectoryEntryCount = 0;
DWORD dataStartSector = 0;

int N = 100;

cout << "Enter the volume name (e.g. C:\\): ";
wcin >> volume;


if (!GetDiskFreeSpaceEx(volume, NULL, &totalBytes, &freeBytes))
{
    std::cout<< "Error: Failed to retrieve volume information\n";
    return 1;
}

if (!GetDiskFreeSpace(volume, &sectorsPerCluster, &bytesPerSector, &numberOfFATs, &sectorsPerFAT))
{
    std::cout<< "Error: Failed to retrieve volume information\n";
    return 1;
}
```

  - ❖ Sử dụng 2 hàm GetDiskFreeSpaceEx và GetDiskFreeSpace thuộc thư viện Windows.h, cung cấp bởi Microsoft Windows API đều được sử dụng để truy xuất thông tin về dung lượng ổ đĩa.

- Hàm GetDiskFreeSpaceEx trả về thông tin chi tiết hơn về dung lượng ổ đĩa, bao gồm tổng số byte trên ổ đĩa, số byte trống khả dụng, kích thước của một cluster, số cluster trên một ổ đĩa, số sector trên một cluster, số cluster được dành riêng cho FAT (File Allocation Table), và số sector được dành riêng cho Root Directory.
- Hàm GetDiskFreeSpace trả về một số thông tin cơ bản về dung lượng ổ đĩa, bao gồm kích thước của một cluster, số cluster trên một ổ đĩa, số sector trên một cluster, số cluster được dành riêng cho FAT (File Allocation Table).
  - ❖ Trong cả hai hàm, tham số đầu tiên lpDirectoryName và lpRootPathName là một chuỗi đại diện cho đường dẫn của ổ

đĩa được truy xuất thông tin. Tham số tiếp theo trả về thông tin về dung lượng của ổ đĩa.

- Mô tả các bước thực hiện:
  - ✓ Bước 1: Tính các chỉ số: số cluster của file, số byte của file Fi, số chữ (trong bài 3 volume đều có kích thước sector là 512 Bytes)
  - ✓ Bước 2: Mở file để ghi.
  - ✓ Kiểm tra cluster

    + Nếu bằng 1: ghi 1 số là đủ chiếm 1 cluster

    + Nếu >= 2: (byte / (4 + 2)) / 2 + 1. Trong đó, 4 + 2 là số bytes cần thiết để lưu mỗi giá trị số nguyên <2020 + i> và 2 bytes kí tự xuống dòng '\n'. byte / (4 + 2): số lượng dòng tối đa có thể ghi vào file sao cho không vượt quá byte của cluster. Sau đó chia 2 để giảm số lượng dòng một nửa để tránh việc vượt quá số byte của cluster (mỗi dòng sẽ chiếm ít nhất 2 byte). Cộng 1: thêm một dòng để đảm bảo chiếm đủ số cluster cần thiết.

    + Đóng file.

II.1.1. Tạo các file .Dat theo tham số đầu vào

- Mô tả chi tiết hàm main của chương trình

  - Theo đề bài yêu cầu tạo 100 file Fi nên ta có biến int N = 100
  - Yêu cầu nhập đường dẫn volume cần tạo file (VD: E:\)

```
int N = 100;

cout << "Enter the volume name (e.g. C:\\): ";
wcin >> volume;
```

  - Tạo một vòng lặp N lần:

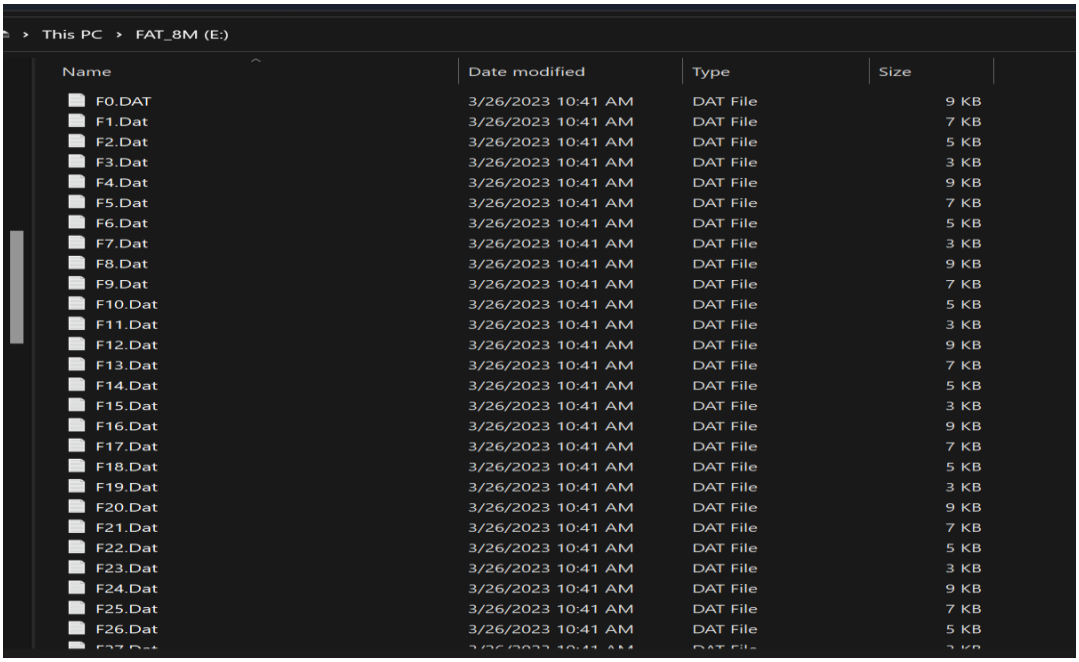
```
    fstream f;
    //convert TCHAR to string
    string volume_path = "";
    for (int i = 0; i < MAX_PATH + 1; i++) {
        if (volume[i] == '\0') {
            break;
        }
        volume_path += volume[i];
    }
    for (int i = 0; i < N; i++) {
        string file_Path = volume_path + "F" + to_string(i) + ".Dat";
        create_DAT_file(file_Path, sectorsPerCluster, bytesPerSector, i);
    }

    return 0;
}
```

+ Xác định đường dẫn file: nối chuỗi đường dẫn đã nhập với kí tự F, giá trị i đã dược chuyển đổi sang chuỗi và chuỗi phần mở rộng .Dat (VD: E:\F0.Dat, E:\F1.Dat,…)

+ Gọi hàm Create_DAT_file với tham số file_name, sectorsPerCluster (số sectors mỗi cluster), số bytes mỗi Sector, và i đã có biến chạy vòng lặp).

- Sau khi chương trình chạy sẽ trả về 100 file Fi trong volume đã chọn.



II.2. Câu 1 – B: Dự đoán số cluster của RDET trên volume FAT32

- Cluster bắt đầu của RDET được lưu bằng số nguyên 4 byte ở offset 2Ch trên Boot sector: Các byte 2C 2D 2E 2F: 02 00 00 00 (LE) nên số cluster bắt đầu là 00000002h = 2

- N = 100, từ file F0 -> F99

=> Số cluster = 2(bắt đầu) + 4(F0) + 3(F1) + 2(F2) + 1(F3) + 4(F4) +… 2(F98) + 1(F99)

= 2 + 4.(25) + 3.(25) + 2(25) + 1(25) = 250

II.3. Câu 1 – C: Xóa các file F0, F2, F4,…, F2n.dat bằng cách tạo và chạy một file .BAT

- Cách xóa các file F2n.Dat bằng việc tạo một đoạn mã lệnh viết bằng ngôn ngữ batch script trên hệ điều hành Windows, được sử dụng để xóa các tệp tin có tên F2n.Dat (xóa các tập tin có số thự tự chẵn).

```
@echo off

set /a n=%1
for /l %%i in (0,2,%n%) do (
    del F%%i.Dat
)

echo Done
```

  - Dòng lệnh đầu tiên "@echo off" sẽ tắt việc hiển thị các thông báo đang được thực thi trên màn hình console.
  - Dòng lệnh tiếp theo "set /a n=%1" sẽ lấy đối số đầu tiên được truyền vào lệnh từ dòng lệnh lệnh command prompt và gán cho biến "n".
  - Dòng lệnh thứ ba "for /l %%i in (0,2,%n%) do (del F%%i.Dat)" sử dụng vòng lặp "for" để xóa các tệp tin có tên F0.Dat, F2.Dat, F4.Dat, và cứ tiếp tục cho đến khi đến tệp tin có tên Fn.Dat. Cụ thể, vòng lặp sẽ lặp lại với biến "%%i" bắt đầu từ 0, tăng lên 2 sau mỗi lần lặp cho đến khi giá trị của biến "%%i" đạt đến giá trị "n". Trong mỗi lần lặp, lệnh "del F%%i.Dat" sẽ được thực thi để xóa tệp tin có tên F%%i.Dat.

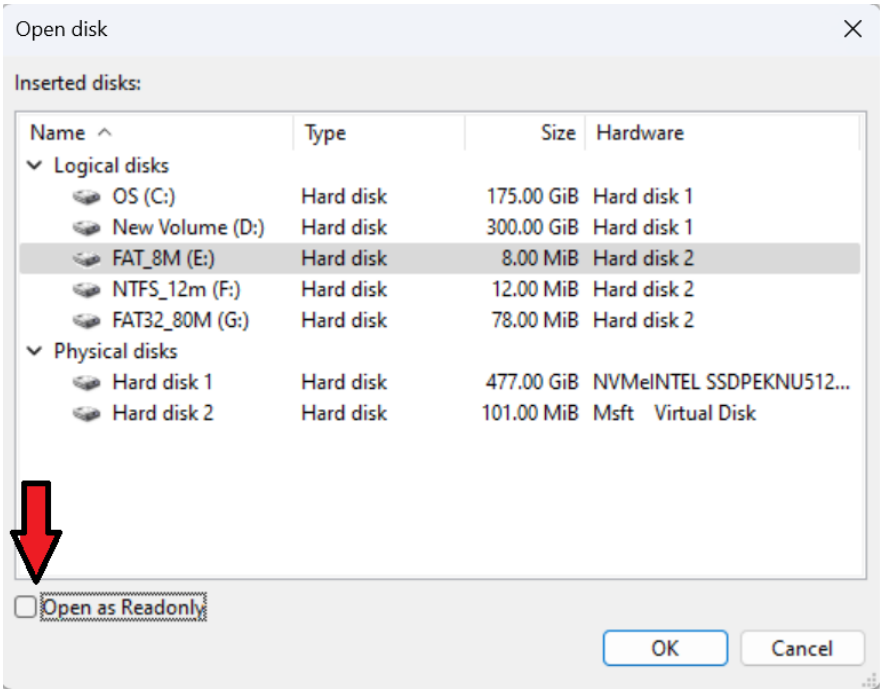
- Dòng lệnh cuối cùng "echo Done" được sử dụng để hiển thị thông báo "Done" trên màn hình console khi quá trình xóa tệp tin hoàn tất. Nếu không có tệp tin cần xóa sẽ thông báo "The system cannot find the file specified."
  VD: cần xóa file chẵn từ F0 cho đến F50

```
D:\>delete_files.bat 50
```

  *Lưu ý: cần bổ sung thêm đường dẫn volume ở dòng del F%%i.Dat. VD cần xóa các file có thứ tự chẵn trong volume E thì ta cần thêm vào như sau: del E:\F%%i.Dat

II.4. Câu 1 – D: Cứu lại F0.dat trên volume FAT bằng cách dùng HexEditor (Phần mềm HxD)

- Mở ổ đĩa(volume) ảo lên bằng HexEdit và bỏ chọn chế độ readonly



- Từ bootsector của ổ đĩa, ta lần lượt lấy các thông số bootsector (S_B), FAT (SF), số lượng FAT(NF), số sector trên 1 cluster (SC), cluster bắt đầu RDET (SRDET).

Thông số	Vị trí	BE – Byte	LE - Byte	Kích thước (sector)
S_B	0xE – 0xF	0x0400	0x0004	4
S_F	0x16 – 0x17	0x0600	0x0006	6
N_F	0x10	0x02	0x02	2 (bảng)
S_C	0x0D	0x08	0x08	8

- Vị trí cluster thứ 2:

+ Clus2 = Sb + Sf*Nf + (2 - 2)*Sc = 4 + 6*2 = 16

=> Như vậy, ta biết được vị trí của bảng RDET nằm ở sector 16. Ta di chuyển đến bảng RDET và quan sát các entry của file.

Ta tiến hành đọc thông tin trên bảng entry của RDET, tìm kiếm thông tin của file F0.Dat:

```
00002050  20 00 56 00 6F 00 6C 00 75 00 00 00 6D 00 65 00   .V.o.l.u...m.e.
00002060  53 59 53 54 45 4D 7E 31 20 20 20 16 00 1A 22 55  SYSTEM~1   ..."U
00002070  7A 56 7A 56 00 00 23 55 7A 56 02 00 00 00 00 00  zVzV..#UzV......
00002080  E5 46 00 30 00 2E 00 44 00 61 00 0F 00 95 74 00  åF.0...D.a...•t.
00002090  00 00 FF FF FF FF FF FF FF FF 00 00 FF FF FF FF  ..ÿÿÿÿÿÿÿÿ..ÿÿÿÿ
000020A0  E5 30 20 20 20 20 20 20 44 41 54 20 00 78 27 55  å0      DAT .x'U
000020B0  7A 56 7A 56 00 00 28 55 7A 56 04 00 04 20 00 00  zVzV..(UzV... ..
```

Thông số	Offset	BE - Byte	LE- Byte	Chuyển đổi
Tên	0			åF0
Phần mở rộng	8			D A T
Vị trí cluster	1A	0x0400	0x0004	4d

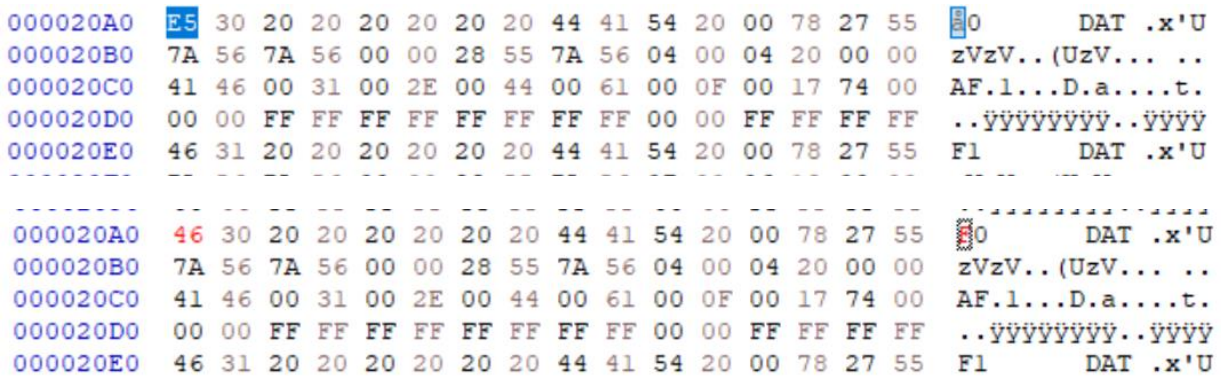
Vậy file F0.Dat nằm ở cluster 4, ta xác định vị trí như sau:

+ $Clus_4$ = Sb + Sf*Nf + (8 - 2)*Sc = 4 + 6 * 2 + 6 * 8 = 64

=> Ta đến sector 64 để kiểm tra nội dung của tập tin F0.Dat

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00008000   32 30 32 30 0D 0A 32 30 32 30 0D 0A 32 30 32 30  2020..2020..2020
00008010   0D 0A 32 30 32 30 0D 0A 32 30 32 30 0D 0A 32 30  ..2020..2020..20
00008020   32 30 0D 0A 32 30 32 30 0D 0A 32 30 32 30 0D 0A  20..2020..2020..
00008030   32 30 32 30 0D 0A 32 30 32 30 0D 0A 32 30 32 30  2020..2020..2020
00008040   0D 0A 32 30 32 30 0D 0A 32 30 32 30 0D 0A 32 30  ..2020..2020..20
00008050   32 30 0D 0A 32 30 32 30 0D 0A 32 30 32 30 0D 0A  20..2020..2020..
00008060   32 30 32 30 0D 0A 32 30 32 30 0D 0A 32 30 32 30  2020..2020..2020
00008070   0D 0A 32 30 32 30 0D 0A 32 30 32 30 0D 0A 32 30  ..2020..2020..20
00008080   32 30 0D 0A 32 30 32 30 0D 0A 32 30 32 30 0D 0A  20..2020..2020..
00008090   32 30 32 30 0D 0A 32 30 32 30 0D 0A 32 30 32 30  2020..2020..2020
000080A0   0D 0A 32 30 32 30 0D 0A 32 30 32 30 0D 0A 32 30  ..2020..2020..20
000080B0   32 30 0D 0A 32 30 32 30 0D 0A 32 30 32 30 0D 0A  20..2020..2020..
000080C0   32 30 32 30 0D 0A 32 30 32 30 0D 0A 32 30 32 30  2020..2020..2020
000080D0   0D 0A 32 30 32 30 0D 0A 32 30 32 30 0D 0A 32 30  ..2020..2020..20
000080E0   32 30 0D 0A 32 30 32 30 0D 0A 32 30 32 30 0D 0A  20..2020..2020..
```

- Quan sát thấy nội dung là của F0.Dat, ta quay lại bảng RDET, đến entry của file F0.Dat và sửa byte đầu thành 0x46 (kí tự F ở mã ASCII)



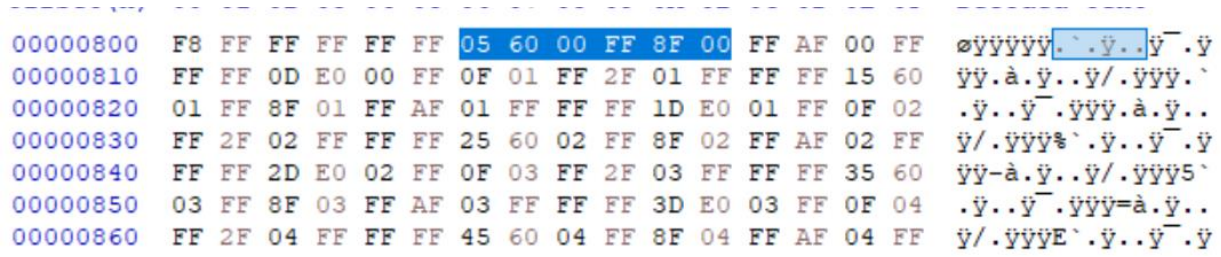
Ta xem kích thước file F0.Dat, từ Offset 1C (lấy 4 byte): 0x04200000 (BE-Byte) => đảo ngược lại ta được 0x2004 bằng 8196 Byte

Mà mỗi cluster chiếm 4096B, suy ra F0.Dat chiếm: 8196 / 4096 = 2,0009765 ≈ 3 clusters. Tiếp theo, ta di chuyển về Bảng FAT để cập nhập thông tin cluster thứ 4 đến 6 cho tập tin F0.Dat (3 cluster của file F0.Dat).

- Vị trí bảng FAT nằm sau Bootsector, $S_B = 4$. Vậy ta đến sector 4:

- Đoạn mã 0xFFFFFFF8 là dãy byte bắt đầu bảng FAT.

- Volume định dạng FAT12 nên 1,5 byte sẽ đánh dấu là 1 cluster: ta điền vị trí 3 cluster tiếp theo của file F0.Dat vào các vị trí còn trống.



- Save lại và ta đã khôi phục được file F0.Dat

III. Câu 2

III.1. Câu 2 – A: Xây dựng mô hình và thiết kế kiến trúc tổ chức cho một hệ thống tập tin được chứa trong file

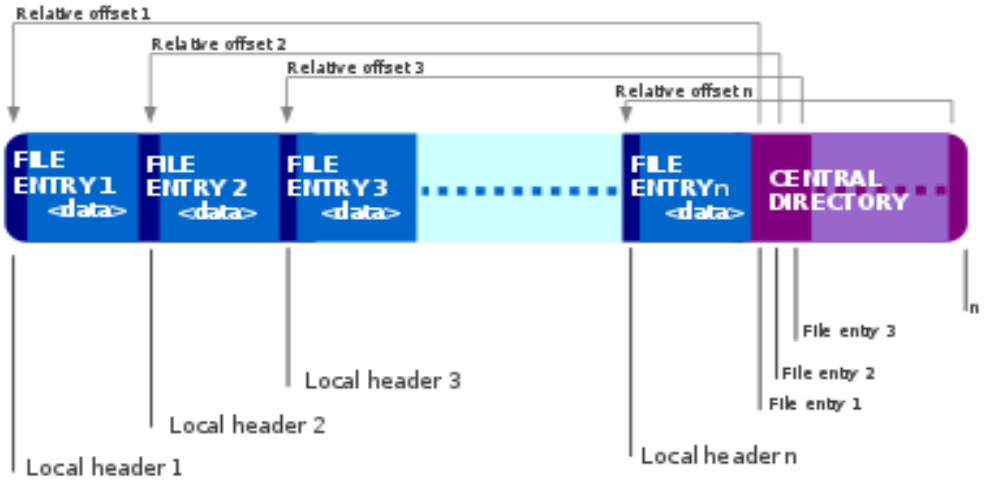
III.1.1. Thiết kế

- Nhóm em đề xuất hai phiên bản cấu trúc lưu trữ dữ liệu khác nhau được sử dụng để quản lý dữ liệu trên một phương tiện lưu trữ nào đó.

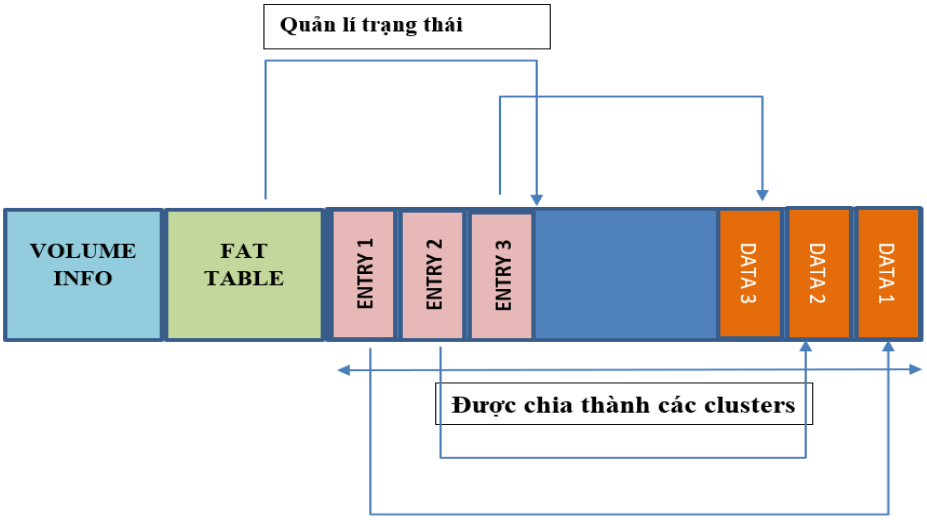
- Phiên bản 1: bao gồm 4 vùng lưu trữ bao gồm: Volume Info, Fat Table, Entry Table và Data. Phiên bản 1 bao gồm 4 vùng lưu trữ bao gồm Volume Info, Fat Table, Entry Table và Data. Phiên bản này được thiết kế dựa trên cấu trúc file zip(hình 1) và bảng Fat trong cấu trúc Fat, với Entry i sẽ lưu thông tin của vùng Data i tương ứng.

  * Tuy nhiên, cấu trúc này có một hạn chế đó là Entry Table sẽ được định sẵn kích thước, do đó khi Entry Table đầy mà Data vẫn còn chỗ, không thể nhập thêm dữ liệu vào Volume.
- Phiên bản 2 (hình 2): sử dụng một cách tiếp cận khác, thay vì sử dụng một vùng Data riêng biệt, nó được thêm vào Volume ngược chiều với Entry.

  => Ưu điểm: Điều này cho phép Data và Entry sử dụng chung bộ nhớ và không có sự phân chia rõ ràng ranh giới giữa chúng như phiên bản 1, cải thiện khả năng quản lí dữ liệu trên Volume. Trong quá trình thiết kế, nhóm em đã chọn sử dụng cấu trúc này để lữu trữ dữ liệu.



Hình 1 (Source:https://en.wikipedia.org/wiki/ZIP_%28file_format%29)



Hình 2

- Volume Information:

  - Bảng dưới đây mô tả các trường trong volume

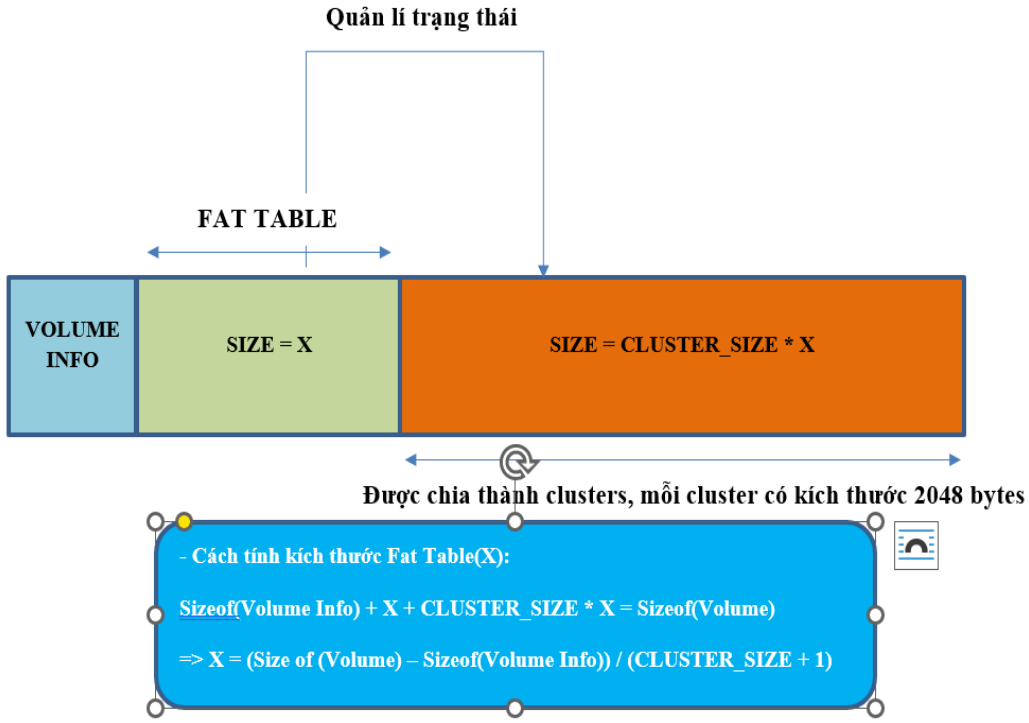
Offset	Kích thước (byte)	Mô tả
0	4	Thể hiện signature của volume
4	4	Số lượng entry ở thời điểm hiện tại
8	4	Tổng kích thước của các entry hiện tại trong volume
12	8	Kích thước của volume
20	2	Độ dài mật khẩu volume
22	n	Mật khẩu volume với độ dài được lưu trữ ở trên

- Fat Table: sẽ quản lí cluster thứ i
  - Fat Table thứ i có thể nhận các giá trị:
    + 0: cluster trống
    + 1: cluster đang được sử dụng
    + 2: cluster đã từng được sử dụng

  => Việc đánh dấu cluster đã từng được sử dụng là số 2 trong Fat Table có mục đích để biết được các cluster nào đã được sử dụng trước đó và không còn khả năng sử dụng lại được nữa. Việc đánh dấu này giúp cho hệ thống file tránh được tình trạng ghi đè lên các dữ liệu đã từng được lưu trữ ở các

cluster đó và đảm bảo tính toàn vẹn của dữ liệu. Ngoài ra, việc đánh dấu cluster đã từng được sử dụng là số 2 cũng giúp cho hệ thống file có thể sử dụng các cluster này để lưu trữ các dữ liệu mới nếu cần thiết, tránh việc phải sử dụng các cluster mới và dẫn đến lãng phí không gian lưu trữ.

* Cách tính kích thước của Fat Table:



- Entry

Offset	Kích thước	Mô tả
0	4	Kích thước dữ liệu lưu
4	2	Chiều dài tên (m)
6	2	Chiều dài mật khẩu (n)
8	8	Vị trí cluster bắt đầu của vùng data
16	2	Chiều dài đường dẫn dữ liệu import (p)
18	2	Số entry con (nếu là file thì nhận giá trị 0)
20	8	Vị trí cluster entry chiếm
28	1	Kiểm tra dữ liệu dang lưu là folder hay file

29	1	Kiểm tra xem dữ liệu lưu đã bị xóa hay chưa
30	m	Tên lưu trong volume
30 + m	n	Mật khẩu sau hash
30 + n + p	P	Đường dẫn dữ liệu import

III.1.2. Thiết kế đưa ra đáp ứng các tiêu chí về bảo mật thông tin, an toàn dữ liệu và có thể phục hồi lại các tập tin đã xóa

- Mật khẩu được đặt sẽ được lưu ở dạng bcrypt
- Lí do mật khẩu được lưu dưới dạng bcrypt

+ Mật khẩu là một thông tin quan trọng để đảm bảo bảo mật tài khoản và dữ liệu. Tuy nhiên, để lưu trữ mật khẩu một cách an toàn và hiệu quả, các phương pháp truyền thống như lưu trực tiếp hoặc lưu dưới dạng encryption đã được chứng minh là không đủ an toàn.

+ Phương pháp hash, trong đó mật khẩu được hash trước khi lưu trữ, là một cách tiếp cận tốt hơn để bảo vệ mật khẩu. Tuy nhiên, một vấn đề với phương pháp này đó là các giá trị đầu vào giống nhau sẽ cho cùng một kết quả đầu ra, dẫn đến khả năng tấn công brute-force. Để khắc phục vấn đề này, phương pháp hash + salt được sử dụng, trong đó một chuỗi salt được thêm vào trước khi hash mật khẩu.

+ Việc sử dụng hash + salt tăng đáng kể độ khó để tấn công brute-force, vì kẻ tấn công phải tìm kiếm một mật khẩu khác mà có cùng salt và hash với mật khẩu gốc. Tuy nhiên, với sức mạnh tính toán của máy tính hiện đại, việc thực hiện hàng triệu lần hash/giây không phải là điều khó khăn. Do đó, để đảm bảo an toàn cho dữ liệu quan trọng, cần phải sử dụng các cách thức bảo vệ khác như sử dụng các thuật toán hash mạnh và sử dụng cơ chế chống tấn công brute-force như đăng nhập sai quá nhiều lần sẽ bị khóa tạm thời.

=> Và đây là những lí do nhóm em chọn bcrypt. Có thể nói Bcrypt là một thuật toán mã hóa mật khẩu an toàn và được sử dụng phổ biến để bảo vệ mật khẩu trên các hệ thống. Bcrypt sử dụng một phương thức gọi là "salt" để mã hóa mật khẩu, trong đó salt là một chuỗi

ngẫu nhiên được tạo ra mỗi khi mật khẩu được tạo hoặc thay đổi. Salt giúp ngăn chặn các cuộc tấn công bằng cách làm cho các mật khẩu giống nhau trông khác nhau, ngay cả khi chúng được tạo từ cùng một chuỗi ký tự.
Bcrypt cũng sử dụng một hàm hash mạnh, gọi là Blowfish, để mã hóa mật khẩu. Blowfish là một hàm hash rất an toàn và khó bị tấn công bởi các kẻ tấn công thông qua các phương pháp như tấn công từ điển, tấn công bằng vét cạn, tấn công bằng precomputation hoặc tấn công bằng bảng cách băm.
Vì vậy, việc sử dụng bcrypt để lưu trữ mật khẩu là rất an toàn và được khuyến khích để bảo vệ dữ liệu của người dùng.

❖ Tổng kết lại:

+ Nhóm của chúng em đặt mục tiêu đảm bảo tính an toàn cho dữ liệu được lưu trữ trong ứng dụng của chúng tôi. Để đạt được mục tiêu này, chúng em yêu cầu mật khẩu cho tất cả các file và thư mục được nhập vào, bao gồm cả volume. Mật khẩu được mã hóa bằng thuật toán bcrypt hoặc SHA256, tùy thuộc vào cài đặt trong mã nguồn.

+ Chúng em cũng quan tâm đến việc lưu trữ dữ liệu một cách an toàn và hiệu quả. Vì vậy, chúng em không cho phép dữ liệu được lưu trữ ở dạng phân mảnh, mà yêu cầu dữ liệu phải được lưu trữ trong các cluster liền kề. Mặc dù cách làm này có thể gây ra một số khó khăn trong việc quản lý bộ nhớ, nhưng chúng em tin rằng nó sẽ giúp bảo vệ dữ liệu của người dùng tốt hơn.

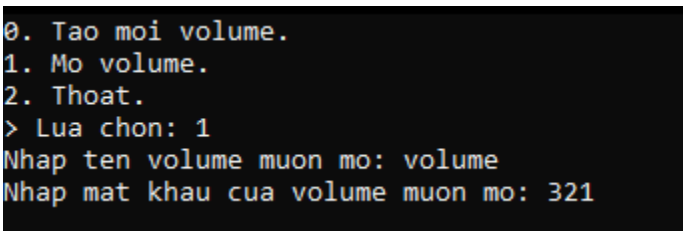
+ Chúng em cũng đã cải thiện tốc độ truy xuất dữ liệu bằng cách lưu trữ các cluster kế tiếp nhau và sử dụng bảng Fat để quản lí trạng thái của các cluster.

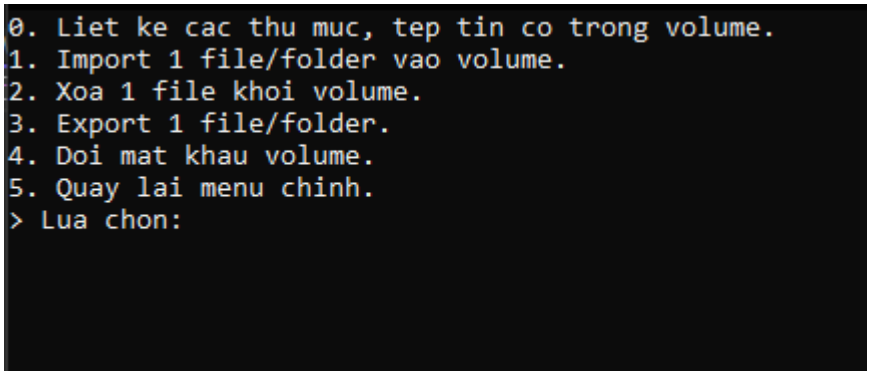
+ Ngoài ra, chúng em cũng nhận thấy rằng tính năng khôi phục file đã xóa của chúng tôi vẫn còn hạn chế và cần được cải thiện.

III.2. Câu 2 – B: Mô tả chương trình thực hiện các chức năng (đoạn code chính, màn hình / giao diện và kết quả chạy chương trình)

III.2.1. Menu chính

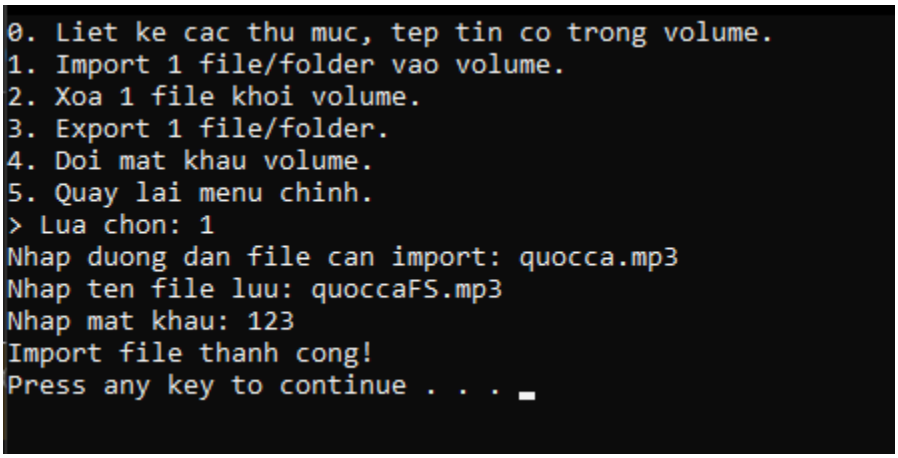
- Menu chính sẽ cung cấp cho người dùng lựa chọn các chức năng như: tạo/định dạng volume, thiết lập/đổi/kiểm tra mật khẩu truy xuất, liệt kê danh sách các tập tin, đặt/đổi mật khẩu truy xuất cho một tập tin, import và export tệp, và xóa tập tin.





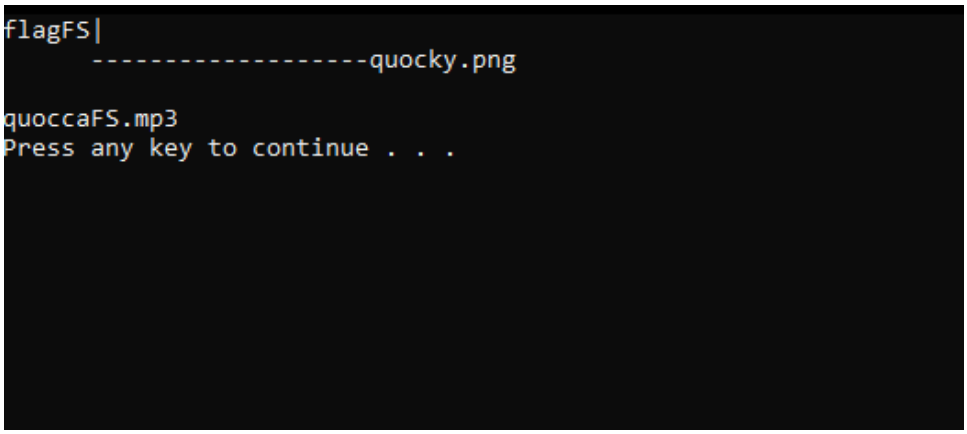
III.2.1. Import file

- Chức năng này cho phép người dùng chọn một tập tin từ bên ngoài và chép (import. Tập tin được chọn sẽ được chuyển vào vùng Data của volume và các thông tin quản lý liên quan sẽ được lưu trong vùng Info.



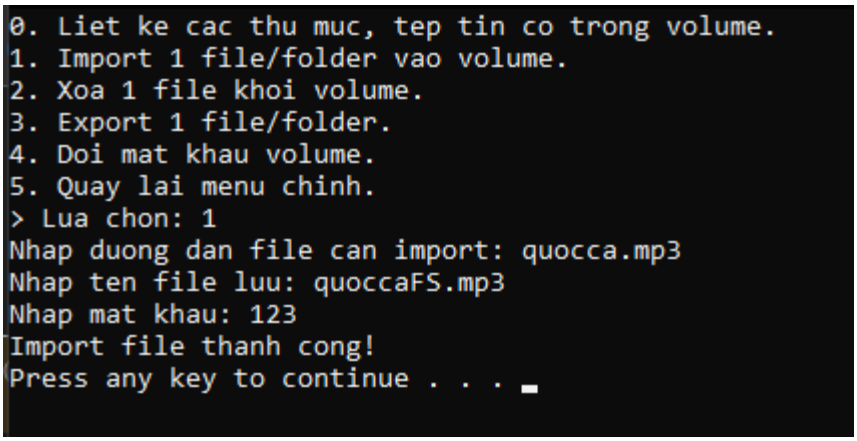
III.2.2. Xem nội dung volume

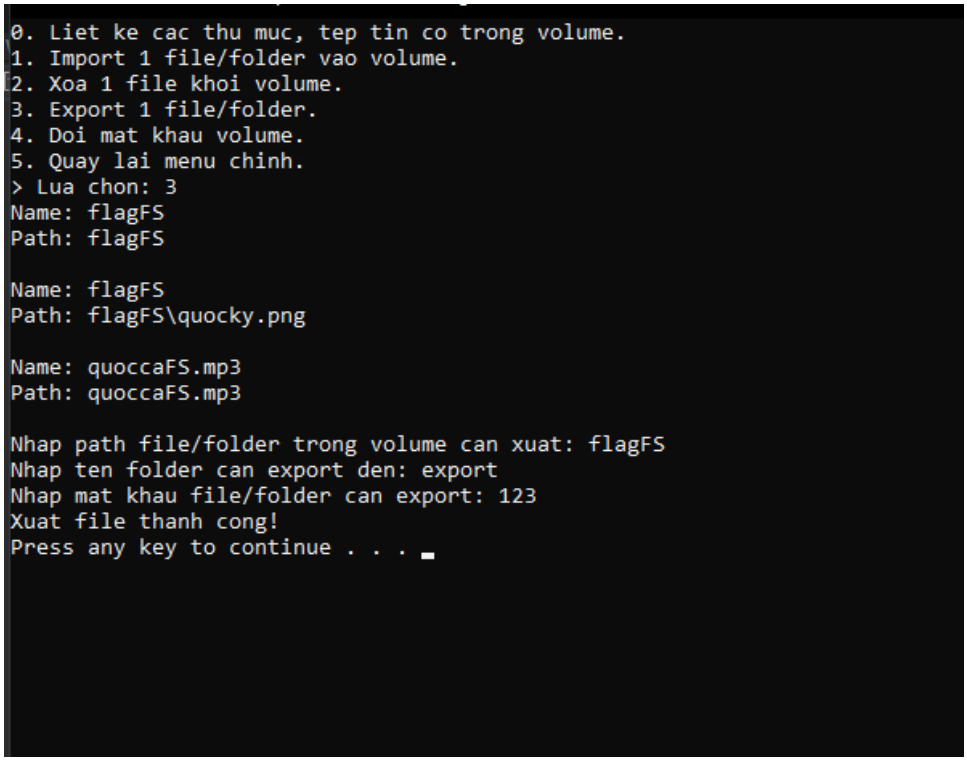
- Chức năng này sẽ liệt kê danh sách các tập tin hiện có trong volume



III.2.3. Import/ Export file

Chức năng này cho phép người dùng chọn một tập tin để chép vào volume, hoặc chọn một tập tin và chép ra bên ngoài. Người dùng sẽ nhập mật khẩu truy xuất. Sau đó, tập tin sẽ được chép vào trong volume với nội dung giống với tập tin được import, hoặc xuất ra tập tin đã có sẵn trong volume.





III.2.4. Đổi mật khẩu volume

Chức năng này cho phép người dùng thiết lập, đổi hoặc kiểm tra mật khẩu truy xuất volume và tập tin. Mật khẩu này sẽ được yêu cầu khi truy cập volume và tập tin, giúp bảo mật thông tin và an toàn dữ liệu.

```
0. Liet ke cac thu muc, tep tin co trong volume.
1. Import 1 file/folder vao volume.
2. Xoa 1 file khoi volume.
3. Export 1 file/folder.
4. Doi mat khau volume.
5. Quay lai menu chinh.
> Lua chon: 4
Nhap mat khau cu cua volume: 123
Nhap mat khau moi cho volume: 321
Mat khau da duoc doi.
Press any key to continue . . .
```

III.2.5. Một số đoạn code

- Định dạng của một volume

```
class Volume
{
private:
    uint32_t _Signature;
    uint32_t _SizeEntryTable;
    uint32_t _NumberOfEntry;
    uint64_t _OffsetEntryTable;
    uint64_t _VolumeSize;
    uint16_t volPassSize;
    string volPass;
public:
    Volume() {};
    Volume(uint32_t volumeSize);
    uint32_t Signature() { return _Signature; }
    uint32_t SizeEntryTable() { return _SizeEntryTable; }
    void SetSizeEntryTable(uint32_t value) { _SizeEntryTable = value; }
    uint32_t NumberOfEntry() { return _NumberOfEntry; }
    void SetNumberOfEntry(uint32_t value) { _NumberOfEntry = value; }
    uint64_t VolumeSize() { return _VolumeSize; }
    void SetNVolumeSize(uint32_t value) { _VolumeSize = value; }
    void SetVolPass(string pw) { volPass = pw; volPassSize = volPass.size(); }
    uint16_t volPassLen() { return volPassSize; }
    uint16_t volumeSize() { return 30 + volPassSize; }
    bool checkPassword(string pw);
    void writeVolInfo(FILE* f);
    void readVolInfo(FILE* f);
    void UpdateVol(FILE* f);
};
```

- Định dạng của một entry (bao gồm file và folder, trong đó folder thì có các sub entry)

```
Entry::Entry()
{
    this->_SizeData = 0;
    this->_NameLen = 4;
    this->_PasswordLen = 3;
    this->_OffsetData = 0;
    this->_FatPosition = 0;
    this->_IsDeleted = false;
    this->pathLen = 0;
    this->path = "";
    this->subEntrySize = 0;
    this->_IsFolder = false;
    this->_Password = "";
    this->_Name = "";
    this->subEntryList = vector<Entry*>(0);
}
```

IV. Trích dẫn

- https://en.wikipedia.org/wiki/ZIP_%28file_format%29

- https://github.com/kieuconghau/file-system

- https://github.com/stbrumme/hash-library

- https://en.wikipedia.org/wiki/Bcrypt