**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG-HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

---o0o---

Logo

Description automatically generated with medium confidence

**BÁO CÁO**

**PROJECT 1:**

**HỆ THỐNG QUẢN LÝ TẬP TIN TRÊN WINDOWS**

**Thành viên nhóm MSSV**

Đỗ Phan Tuấn Đạt 22127057

Phạm Thành Đạt 22127064

Đỗ Đình Hải 22127095

Lê Hồ Phi Hoàng 22127123

Trần Nguyễn Minh Hoàng 22127131

**Lớp:** 22CLC02

**Môn học:** Hệ điều hành

**Học kỳ:** 2

**Năm học:** 2023-2024

# Mục lục

[I. Mục lục 2](#_Toc161121644)

[II. Môi trường làm việc 4](#_Toc161121645)

[1. Môi trường làm việc: 4](#_Toc161121646)

[2. Ngôn ngữ: 4](#_Toc161121647)

[III. Danh sách file trong source code 4](#_Toc161121648)

[IV. Phân tích, giải thích source code 4](#_Toc161121649)

[1. bootSector.h + readBootSector.cpp 4](#_Toc161121650)

[a) bootSector() 5](#_Toc161121651)

[b) readBootSector 6](#_Toc161121652)

[c) printBootSector 7](#_Toc161121653)

[2. fatTable.h + fatTable.cpp 8](#_Toc161121654)

[a) Constructor 8](#_Toc161121655)

[b) Operator = 9](#_Toc161121656)

[c) getClusters: 9](#_Toc161121657)

[d) listClustersOfEntry 9](#_Toc161121658)

[3. headerAttribute.h + headerAttribute.cpp 10](#_Toc161121659)

[a) Constructor 12](#_Toc161121660)

[b) printInfo 13](#_Toc161121661)

[c) getTypeID, getAttributeSize, getFlafNonResident, getNameLength, getPositionOfAttributeName, getFlag, getAttributeID 13](#_Toc161121662)

[4. StandardInfo.h + StandardInfo.cpp 13](#_Toc161121663)

[a) Constructor 14](#_Toc161121664)

[b) printInfo 16](#_Toc161121665)

[5. ATTRIBUTE\_LIST + ATTRIBUTE\_LIST.cpp 17](#_Toc161121666)

[a) Constructor 19](#_Toc161121667)

[b) print 20](#_Toc161121668)

[c) printInfo 21](#_Toc161121669)

[d) Các hàm getHeader, getRecords, getAttributeSize 21](#_Toc161121670)

[6. DATA.h + DATA.cpp 22](#_Toc161121671)

[a) Constructor: 22](#_Toc161121672)

[b) print 24](#_Toc161121673)

[c) printInfo 24](#_Toc161121674)

[d) Các hàm getter, getHeader, getDataSize và getData 24](#_Toc161121675)

[V. Demo sử dụng chương trình 25](#_Toc161121676)

[1. FAT32 25](#_Toc161121677)

[a) Nội dung ổ đĩa 25](#_Toc161121678)

[b) Sử dụng chương trình để đọc ổ đĩa 26](#_Toc161121679)

[2. NTFS 28](#_Toc161121680)

[a) Nội dung ổ đĩa 28](#_Toc161121681)

[b) Sử dụng chương trình để đọc ổ đĩa 29](#_Toc161121682)

[VI. Đóng góp 31](#_Toc161121683)

[VII. Tài liệu tham khảo 31](#_Toc161121684)

# Môi trường làm việc

## Môi trường làm việc:

Visual Studio Code

## Ngôn ngữ:

C++

# Danh sách file trong source code

* bootSector.h + readBootSector.cpp
* fatTable.h + fatTable.cpp
* headerAttribute.h + headerAttribute.cpp
* StandardInfo.h + StandardInfo.cpp
* ATTRIBUTE\_LIST.h + ATTRIBUTE\_LIST.cpp
* DATA.h + DATA.cpp

# Phân tích, giải thích source code

## bootSector.h + readBootSector.cpp

* Chứa class BootSector và các hàm tương ứng để đọc Boot Sector của hệ thống FAT

A screenshot of a computer program

Description automatically generated

* Lớp BootSector bao gồm các thuộc tính: Sc: số sector trên cluster, Sb: số sector thuộc vùng boot sector, Nf: số lượng bảng FAT, Sv: kích thước volume, Sf: kích thước mỗi bảng FAT, startClusterRDET: cluster bắt đầu của RDET, numOfSectorExtra: sector chứa thông tin phụ, numOfSectorSave: sector chứa bản lưu của Boot sector, typeFAT: loại FAT
* Các phương thức:

### bootSector()

* + Default constructor, set tất cả thông tin thuộc tính bằng 0

### readBootSector

A screenshot of a computer

Description automatically generated

* + Xử lý thông tin của bootSector từ 1 mảng 512 byte thành các dữ liệu tương ứng có thể đọc được
    - Sc (Sectors per cluster) từ offset 0x0D trong boot sector.
    - Sb (Sector of boot sector) từ offsets 0x0E và 0x0F trong boot sector, sử dụng 2 bytes.
    - Nf (Number of FATs) từ offset 0x10 trong boot sector.
    - Sv (Number of sectors in the volume) từ offsets 0x20 đến 0x23 trong boot sector, sử dụng 4 bytes.
    - Sf (Sectors per FAT) từ offsets 0x24 đến 0x27 trong boot sector, sử dụng 4 bytes.
    - startClusterRDET (Start cluster of RDET) từ offsets 0x2C đến 0x2F trong boot sector, sử dụng 4 bytes.
    - numOfSectorExtra (Number of sectors for extra data) từ offsets 0x30 và 0x31 trong boot sector, sử dụng 2 bytes.
    - numOfSectorSave (Number of sectors for save data) từ offsets 0x32 và 0x33 trong boot sector, sử dụng 2 bytes.
    - typeFAT (Type of FAT) từ offsets 0x52 đến 0x59 trong boot sector, sử dụng 8 bytes.
  + Kĩ thuật được dùng trong hàm này là phép dịch bit sang trái. Phép dịch bit sang trái di chuyển giá trị của một biểu thức hoặc một giá trị sang phía trái một số lượng bit cụ thể. Ví dụ:
    - Sb = sector[14] + (sector[15] << 8); Dấu "<< 8" được sử dụng để dịch giá trị của sector[15] sang trái 8 bit. Điều này có nghĩa là giá trị của sector[15] được nhân với 2^8 (256). Sau đó, giá trị của sector[14] được cộng vào kết quả. Cách này được sử dụng để kết hợp giá trị của hai byte liên tiếp trong mảng thành một giá trị 16-bit. Tương tự, phép dịch bit sang trái có thể được sử dụng với số lượng bit khác nhau tùy thuộc vào cách dữ liệu được tổ chức trong boot sector.

### printBootSector

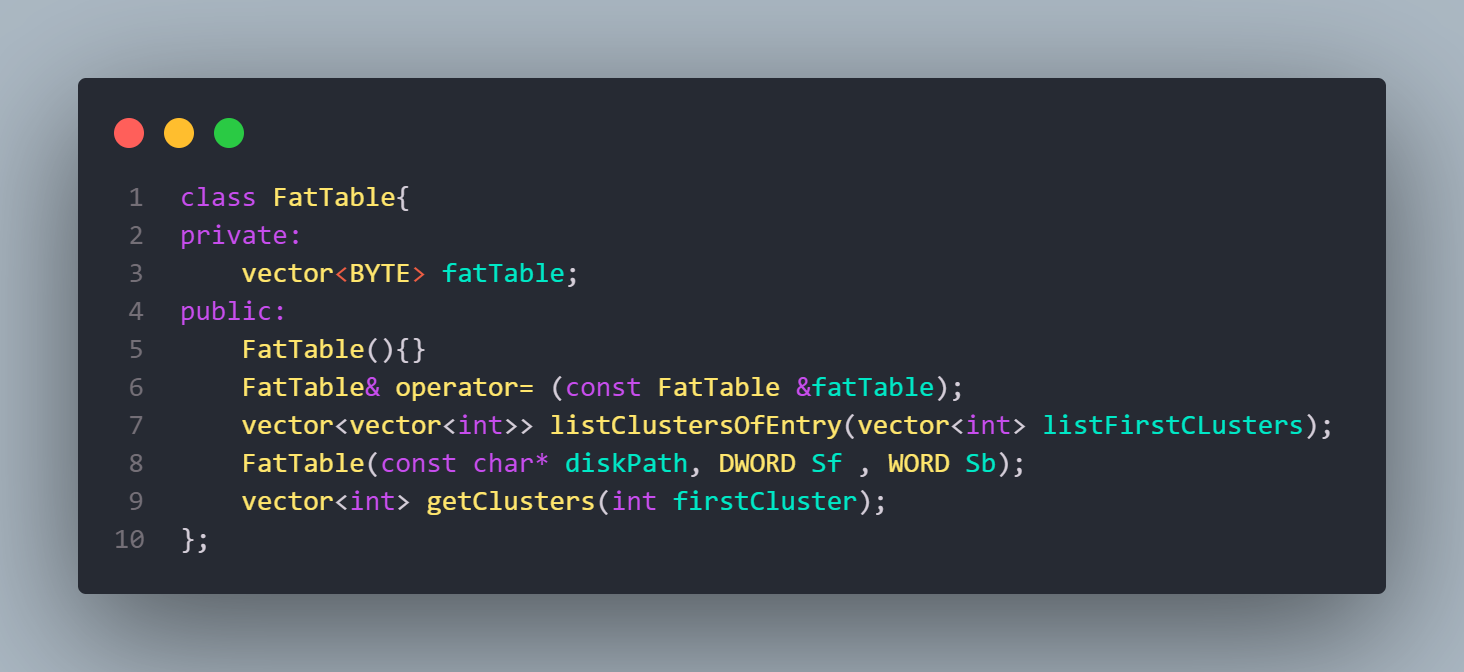
* + In các thông tin đã xử lý ra màn hình

A computer screen shot of a computer code

Description automatically generated

* + Trong hàm có một vòng lặp để chuyển đổi loại FAT từ dạng byte sang dạng string theo phương thức:
    - Khởi tạo một chuỗi rỗng “fatTypeString” để chứa kết quả cuối cùng
    - Vòng lặp for: for (int i = 0; i < 8; ++i) { ... }: Vòng lặp từ 0 đến 7 (8 lần) để xử lý từng byte của giá trị typeFAT.
    - Dịch bit và chuyển đổi:fatTypeString += static\_cast<char>((typeFAT >> (i \* 8)) & 0xFF);:
      * (typeFAT >> (i \* 8)): Dịch bit sang phải i \* 8 vị trí để lấy ra từng byte của typeFAT.
      * & 0xFF: Áp dụng phép AND với 0xFF để giữ lại chỉ 8 bit cuối cùng (0xFF có tất cả các bit ở 8 bit cuối là 1).
      * static\_cast<char>(...): Chuyển đổi giá trị kết quả thành kiểu char để thêm vào chuỗi fatTypeString.
    - Tạo chuỗi kết quả: Mỗi lần lặp, một byte của typeFAT được chuyển đổi thành ký tự và thêm vào chuỗi fatTypeString

## fatTable.h + fatTable.cpp



* Class FatTable có 1 thuộc tính chính là vector chứa byte nội dung của bảng FAT.
* Class FatTable có 4 chức năng:

### Constructor



* + Có 3 tham số:
    - Disk path: tên của ổ đĩa có định dang FAT32
    - Sf: số sector của bảng FAT.
    - Sb: số sector trước bảng FAT.
  + Logic:
    - Đầu tiên tạo 1 mảng có kích thước bằng số sector của bảng FAT với 512 byte 1 sector, Sf \* 512.
    - Sử dụng hàm readSector để đọc các sector này, bỏ qua số byte bằng số sector trước bảng FAT \* 512.

### Operator =

* + So sánh 2 bảng FAT của 2 object FatTable.

### getClusters:



* + Logic: Sử dụng tham số là số thứ tự của cluster đầu tiên và 1 mảng chứa các cluster hư, rỗng hoặc kết thúc. Nếu cluster hiện tại thuộc mảng chứa các byte kết thúc, trả về các cluster đã đọc được. Nếu không, thêm cluster hiện tại vào mảng cluster. Tiếp theo, di chuyển đến cluster tiếp theo và tiếp tục vòng lặp.

### listClustersOfEntry

* + Sử dụng hàm getClusters để lấy tất cả cluster thuộc entry

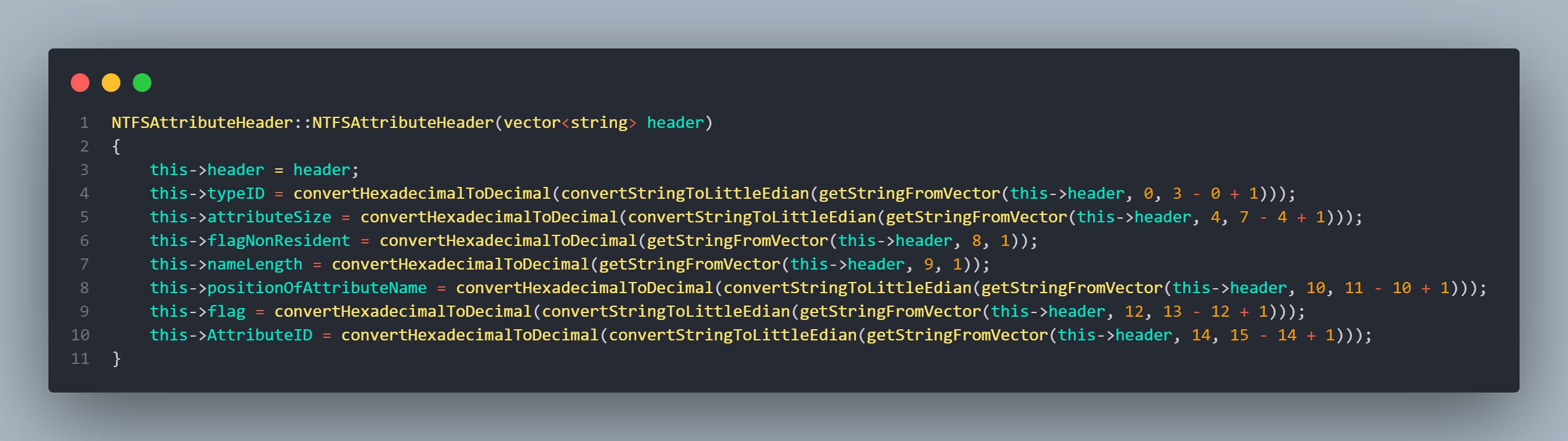
## headerAttribute.h + headerAttribute.cpp

* Chứa class NTFSAttributeHeader, tượng trung cho header của các attribute trong MFT entry



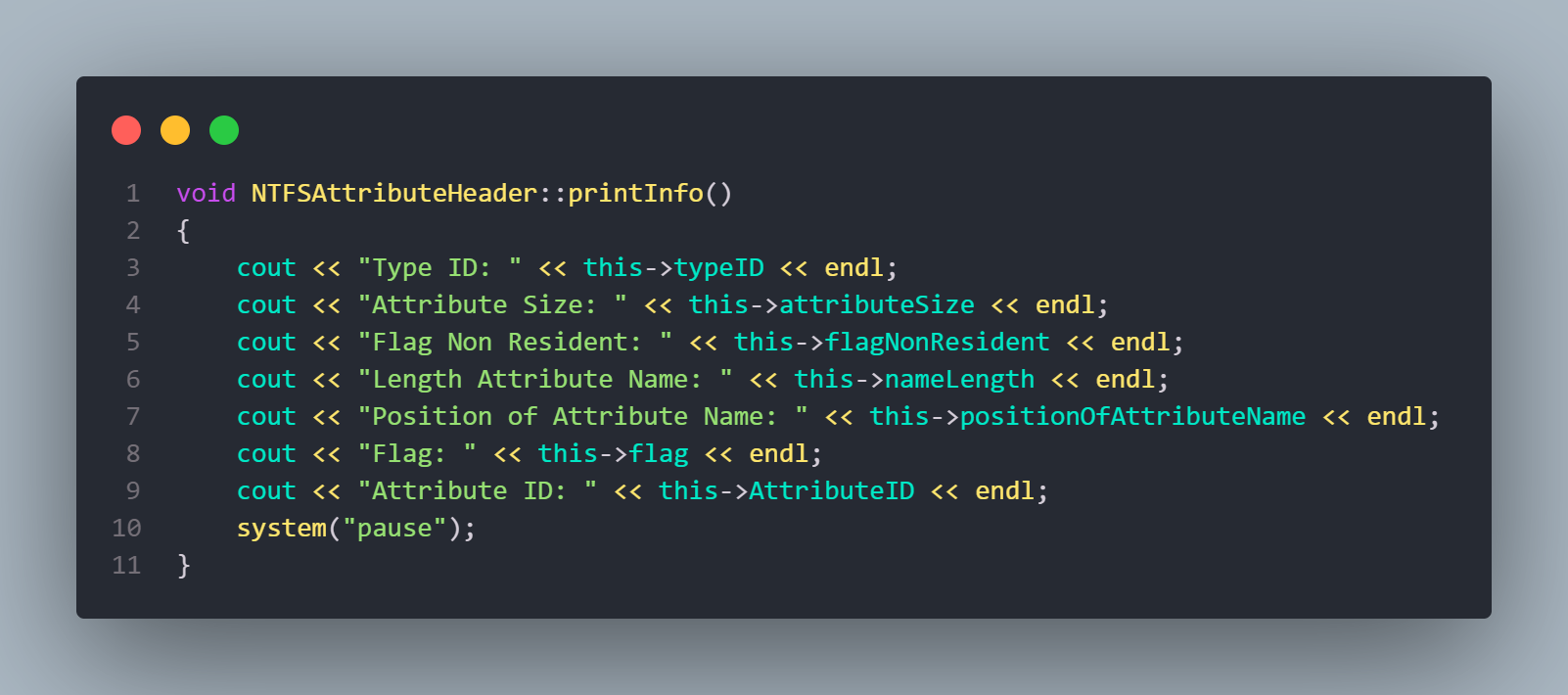
* Class bao gồm các dữ liệu:
  + header: vector lưu lại dữ liệu dạng byte của header
  + Type ID - Mã loại của attribute.
  + Attribute size - Kích thước attribute: số byte mà attribute này chiếm giữ, chiếm 4 byte 4-7.
  + Flag non-resident - Cờ báo loại tệp tin: 1 byte 0x008 báo hiệu loại nội dung là non-resident hay resident.
  + Attribute's name length - Chiều dài tên attribute: 1 byte 0x009 cho biết độ dài của tên attribute.
  + Position of Attribute's name - Vị trí tên của attribute: 2 byte 0x00A và 0X00B cho biết offset của tên chứa attribute
  + Flags - Các cờ báo: 2 byte 0x00C và 0X00D cho biết các cờ báo của attribute.
  + AttributeID - Mã định danh của Attribute: cho biết mã định danh độc nhất của attribute trên toàn entry.
* Class còn bao gồm các hàm:

### Constructor



* + Khởi tạo một đối tượng NTFSAttributeHeader từ dữ liệu header được cung cấp dưới dạng một vector string.
  + this->typeID: Đọc 4 byte đầu tiên từ vector header, chuyển đổi thành dạng little-endian và sau đó chuyển đổi từ hệ thập lục phân sang thập phân để lưu trữ type ID của attribute.
  + this->attributeSize: Đọc 4 byte tiếp theo từ vector header, chuyển đổi thành dạng little-endian và sau đó chuyển đổi từ hệ thập lục phân sang thập phân để lưu trữ kích thước của attribute.
  + this->flagNonResident: Đọc byte tiếp theo từ vector header và chuyển đổi từ hệ thập lục phân sang thập phân để lưu trữ giá trị cờ báo non-resident.
  + this->nameLength: Đọc byte tiếp theo từ vector header và chuyển đổi từ hệ thập lục phân sang thập phân để lưu trữ độ dài của tên attribute.
  + this->positionOfAttributeName: Đọc 2 byte tiếp theo từ vector header, chuyển đổi thành dạng little-endian và sau đó chuyển đổi từ hệ thập lục phân sang thập phân để lưu trữ vị trí bắt đầu của tên attribute trong dữ liệu.
  + this->flag: Đọc byte tiếp theo từ vector header và chuyển đổi từ hệ thập lục phân sang thập phân để lưu trữ giá trị cờ báo của attribute.
  + this->AttributeID: Đọc 2 byte cuối cùng từ vector header, chuyển đổi thành dạng little-endian và sau đó chuyển đổi từ hệ thập lục phân sang thập phân để lưu trữ ID dộc nhất của attribute.

### printInfo



* + In ra dữ liệu đã chuyển đổi của header theo thứ tự: type ID 🡪 attribute size 🡪 non-resident flag 🡪 độ dài tên 🡪 offset đến tên 🡪 cờ báo 🡪 attribute ID

### getTypeID, getAttributeSize, getFlafNonResident, getNameLength, getPositionOfAttributeName, getFlag, getAttributeID

* + Trả về các giá trị tương ứng trong header

## StandardInfo.h + StandardInfo.cpp

* Chứa class StandardInfo, tượng trung cho attribute $STANDARD\_INFORMATION, một attribute quan trọng xuất hiện trong mọi tập tin và thư mục



* Class bao gồm các thuộc tính: flag: giá trị cờ, header: là một object của class NTFSAtrributeHeader, offset: nơi bắt đầu phần nội dung của thuộc tính, vector<string> entry
* Các method:

### Constructor

A black and white screen

Description automatically generated

* + this->entry = entry;: Lưu trữ vector chuỗi entry vào thành viên dữ liệu entry của đối tượng StandardInfo.
  + this->header = NTFSAttributeHeader(entry);: Tạo một đối tượng NTFSAttributeHeader từ dữ liệu entry và lưu trữ nó trong header.
  + this->offset = convertHexadecimalToDecimal(convertStringToLittleEdian(getStringFromVector(this->entry, 20, 21 - 20 + 1)));
    - Sử dụng hàm getStringFromVector để trích xuất một chuỗi từ vector <entry>, lấy giá trị tại byte 20-21.
    - Chuyển đổi chuỗi hex thành số thập phân bằng convertHexadecimalToDecimal.
    - Lưu kết quả vào offset để xác định vị trí bắt đầu phần nội dung
  + this->flag = convertHexadecimalToDecimal(convertStringToLittleEdian(getStringFromVector(entry, this->offset + 32, 35 - 32 + 1)));
    - Lấy giá trị byte 32 đến 35 ở phần nội dung attribute

### printInfo

A screen shot of a computer

Description automatically generated

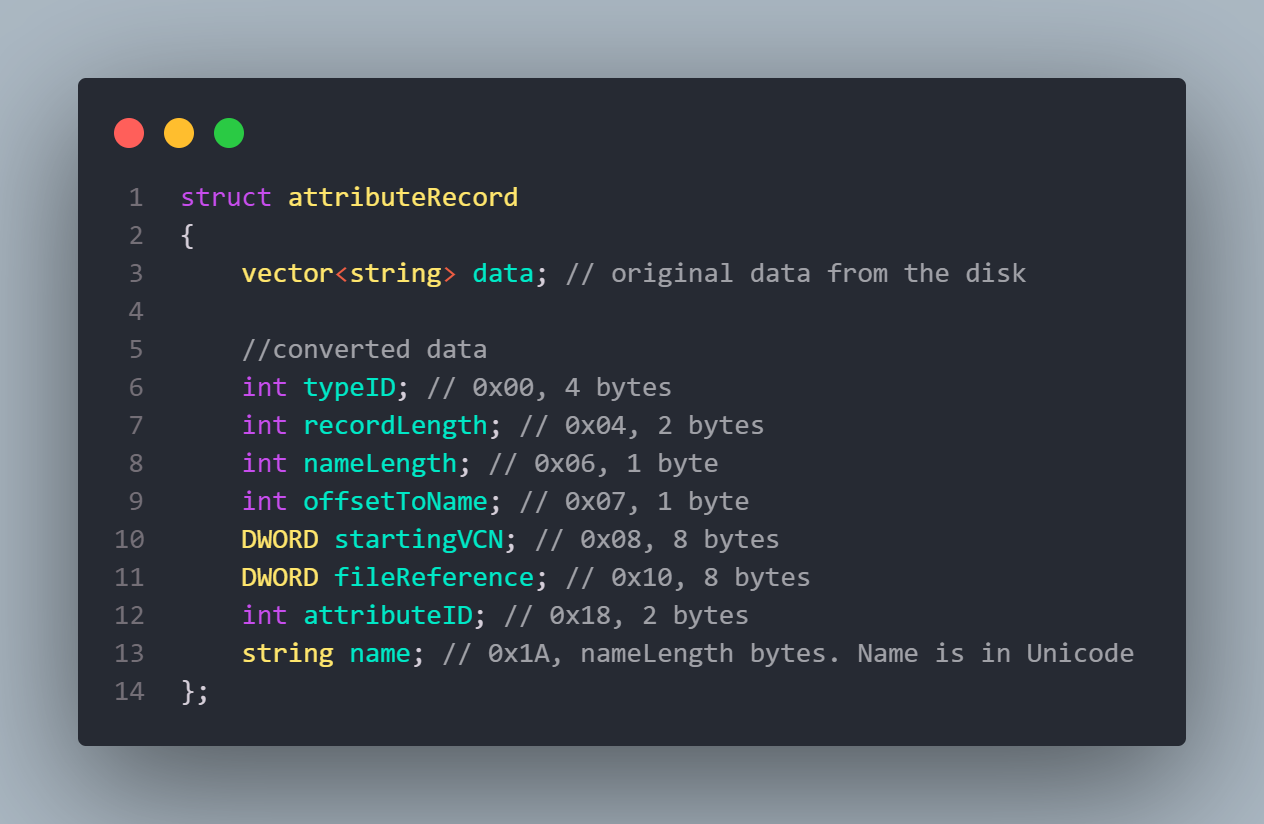
* + Hiển thị thông tin chi tiết đã xử lý từ attribute $STANDARD\_INFORMATION để người dùng có thể đọc và phân tích
  + Sử dụng câu lệnh switch để xác định giá trị của flag và in ra trạng thái tương ứng. Nếu giá trị flag không khớp với bất kỳ trường hợp nào, in ra "Unknown".

## ATTRIBUTE\_LIST + ATTRIBUTE\_LIST.cpp

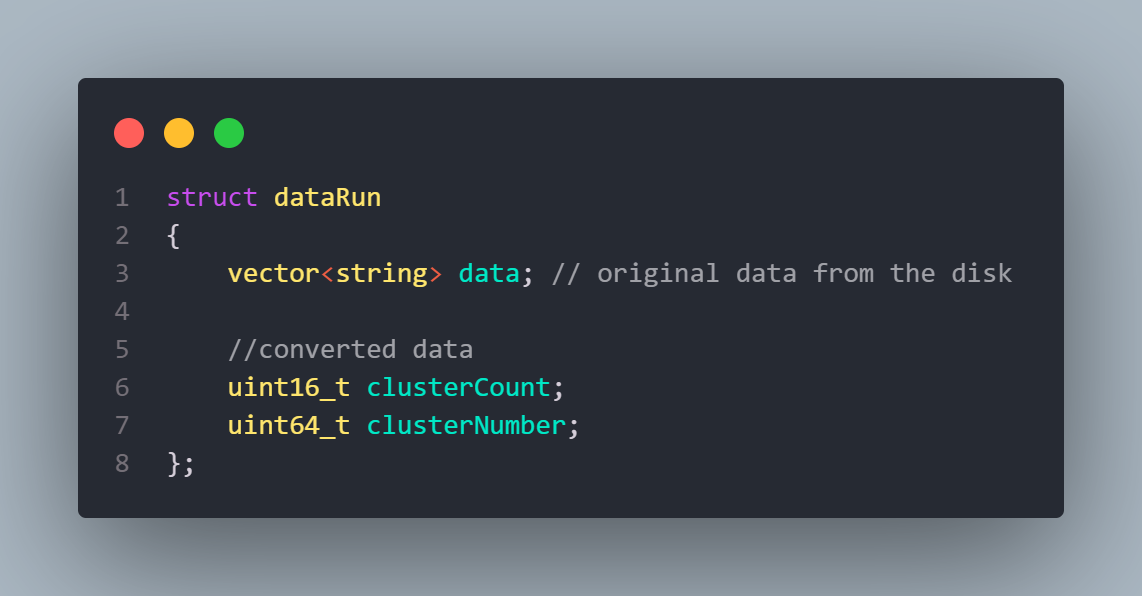
* Chứa class $ATTRIBUTE\_LIST tượng trưng cho attribute $ATTRIBUTE\_LIST trong một entry



* Dữ liệu trong class bao gồm:
  + Dữ liệu gốc dạng byte từ đĩa
  + Header của NTFS attribute
  + Danh sách các thuộc tính theo format:

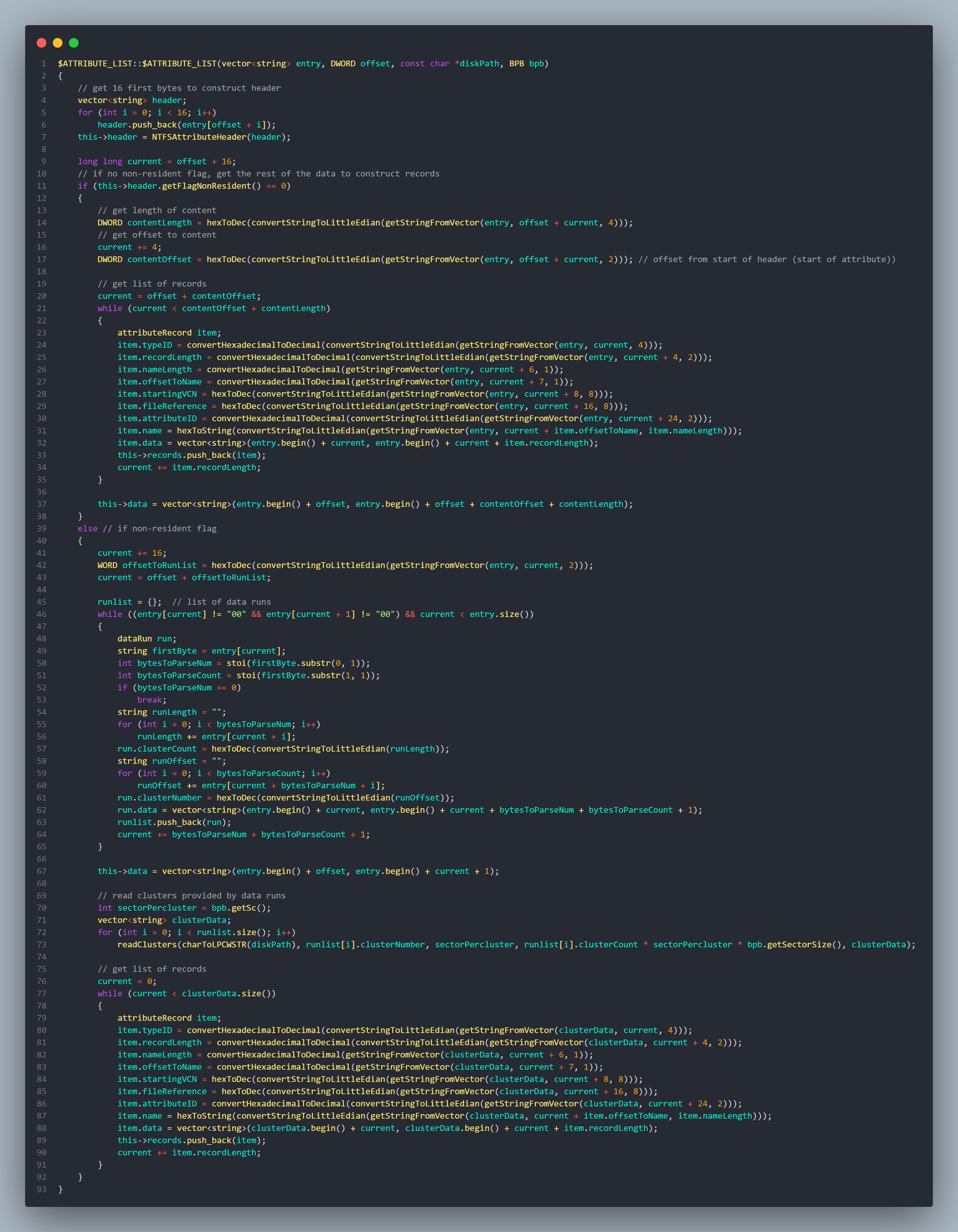


* + Danh sách các data run trong trường hợp có non-resident flag, mỗi data run có format:



* Trong class còn có các function:

### Constructor



* + Dòng 4-7: Khởi tạo một vector header để lưu trữ 16 byte đầu tiên từ dữ liệu đọc được, xây dựng tiêu đề NTFSAttributeHeader.
  + Dòng 11-34: Nếu flagNonResident trong header là 0 (attribute thuộc kiểu resident), chỉ đọc phần còn lại của dữ liệu để xây dựng danh sách các attribute (attributeRecord)
    - Dòng 14-17: Đọc độ dài của nội dung attribute và vị trí của nội dung trong dữ liệu đọc được.
    - Dòng 20-34: Duyệt qua dữ liệu từ vị trí bắt đầu đến kết thúc nội dung thuộc tính, xây dựng các bản ghi attribute và lưu trữ chúng trong vector records.
  + Dòng 39-91: Nếu flagNonResident là 1 (attribute thuộc kiểu non-resident), xây dựng danh sách các data run:
    - Dòng 41-43: Đọc vị trí của danh sách các data run.
    - Dòng 46-65: Duyệt qua danh sách các run dữ liệu và xây dựng các đối tượng dataRun, lưu trữ chúng trong vector runlist bằng vòng lặp:
      * Đọc byte đầu tiên, chia ra thành 2 phần, nửa đầu của byte là số byte cần đọc để lấy chỉ số của cluster cần đến, nửa sau là số byte để lấy số lượng cluster cần đọc
      * Xử lý các byte vừa đọc và lưu vào giá trị tương ứng
      * Điều chỉnh offset để đọc data run tiếp theo, nếu chưa thấy 2 byte liên tiếp có giá trị “00” và chưa hết entry thì quay lại đầu vòng lặp
    - Dòng 70-73: Đọc dữ liệu từ các cluster được cung cấp bởi các data run.
    - Dòng 76-91: Duyệt qua dữ liệu đã đọc từ các cluster và xây dựng các bản ghi attribute, lưu trữ chúng trong vector records.

### print

* + In ra dữ liệu của attribute (dạng byte)

### printInfo



* + In ra dữ liệu đã chuyển đổi của attribute $ATTRIBUTE\_LIST, bao gồm vòng lặp để in ra danh sách các attribute, và danh sách các data run nếu attribute thuộc loại non-resident

### Các hàm getHeader, getRecords, getAttributeSize

* + Để trả về header, danh sách các attribute, hoặc kích thước của attribute $ATTRIBUTE\_LIST
* Trong file còn có các hàm hexToDec và hexToString để hỗ trợ việc xử lý dữ liệu

## DATA.h + DATA.cpp



* Class Data (dùng để biểu diễn attribute $DATA) có 4 thuộc tính chính:
  + Data size: một số nguyên lưu trữ số byte là dữ liệu của tập tin được attribute $DATA lưu trữ. Data size thường nằm ở byte số 16 đến byte số 19 tính từ đầu attribute $DATA.
  + Data offset: là một số nguyên chứa vị trí bắt đầu của phần nội dung. Nếu header của attribute $DATA có cờ resident, data offset sẽ cho biết thứ tự byte bắt đầu của phần nội dung ngay sau attribute $DATA. Ngược lại, nếu có cờ non-resident, data offset sẽ cho biết thứ tự byte bắt đầu của phần nội dung trong runlist. Nếu có là resident, data offset sẽ là 2 byte 20 và 21, còn nếu là non-resident, data offset sẽ là 2 byte 32 và 33.
  + Header: header của attribute $DATA chiếm 16 byte.
  + Data: Một vector byte chứa nội dung đọc được.
* Class Data có 6 chức năng:

### Constructor:

* Tạo 1 attribute $DATA và đọc nội dung của attribute đó



* + 4 tham số được truyền vào là:
    - Diskpath: một chuỗi chứa tên ổ đĩa chứ tệp tin được đọc.
    - Entry: một vector chuỗi chứa nội dung của toàn entry chứa attribute $DATA được đọc.
    - BPB: BIOS PARAMETER CLOCK, một object thuộc class BPB, chứa thông tin mô tả về tổ chức của đĩa logic và hệ thống quản lý tập tin. BPB chứa Sc là số sector của 1 cluster, sẽ dùng trong việc tính toán.
    - Offset: offset từ đầu entry đến vị trí của attribute $DATA.
  + Logic:
    - Đầu tiên, lấy header của attribute $DATA bằng việc lấy 16 byte đầu tiên tính từ offset.
    - Tiếp theo, lấy data size tại byte 16, 17, 18, 19 và data offset tại byte 20, 21.
    - 2 trường hợp loại dữ liệu có thể xảy ra:
      * Nếu dữ liệu thuộc loại resident: Sử dụng 1 vòng lặp có độ dài bằng data size, thu thập số byte bắt đầu từ data offset tính từ đầu attribute $DATA.
      * Nếu dữ liệu thuộc loại non-resident:
        + Cần lấy được runlist. Di con trỏ đến byte 32 và 33 để lấy offset đến runlist.
        + Sau đó di con trỏ đến vị trí bắt đầu runlist
        + Sử dụng vòng lặp while đến khi không có 2 byte liên tiếp nào bằng 0, có nghĩa là runlist vẫn còn dữ liệu
        + Sử dụng cấu trúc dataRun, có thuộc tính data là dữ liệu được lấy, clusterCount là số cluster thuộc runlist và clusterNumber là số thứ tự của cluster.
        + Lấy byte đầu tiên của runlist, chia làm 2 phần

Phần đầu là số byte chứa số thứ tự của cluster.

Phần sau là số byte chứa số cluster cần lấy.

* + - * + Với số byte chứa số thứ tự của cluster và số byte chứa số cluster cần lấy sẽ được số thứ tự và số cluster cần lấy.
        + Sử dụng hàm readCluster để đọc nội dung cluster vào 1 vector string.
        + Cuối cùng, thu thập số byte từ vector cluster đó bằng đúng data size.

### print

* + In header và vector data dưới dạng BYTE.

### printInfo

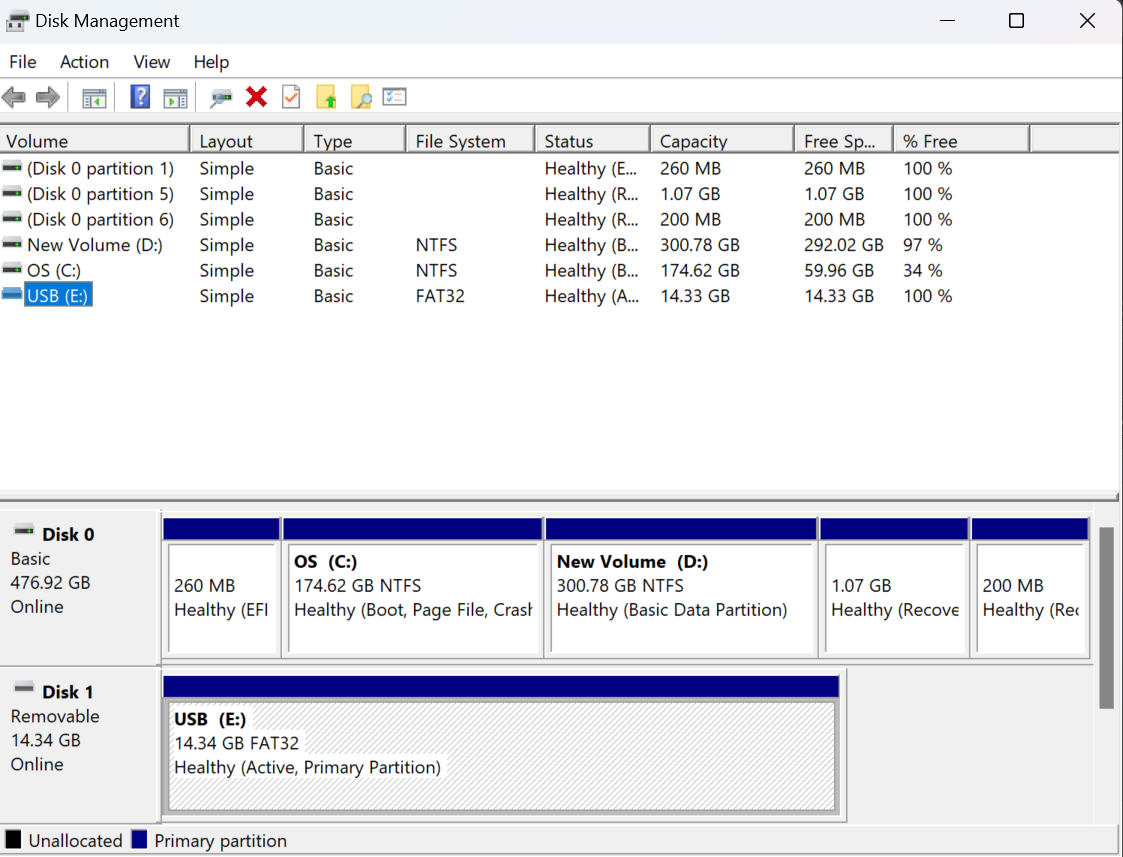
* + In header và vector data dưới dạng ký tự ASCII.

### Các hàm getter, getHeader, getDataSize và getData

* + Lấy thuộc tính header, size của attribute và data từ class DATA

# Demo sử dụng chương trình

## FAT32



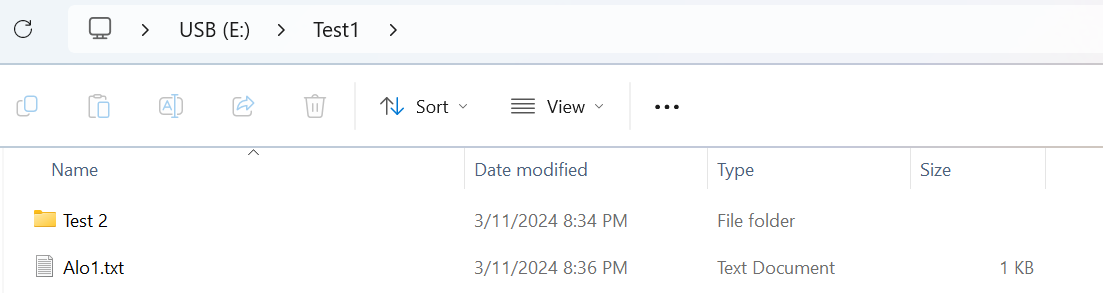
Ổ đĩa được chọn trong demo sẽ là ổ E

### Nội dung ổ đĩa

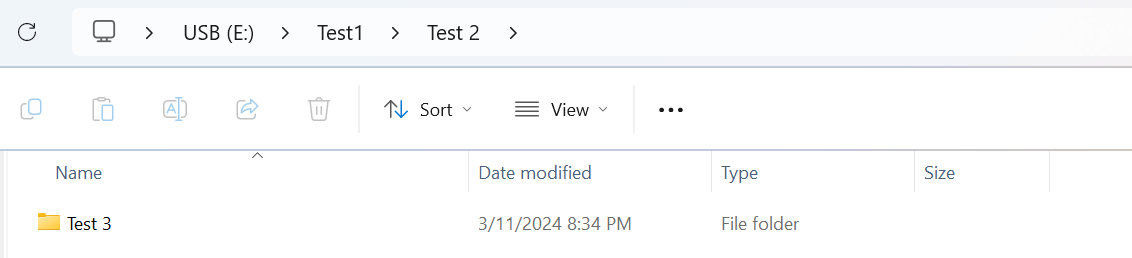
* Trong ổ E sẽ có 1 thư mục duy nhất



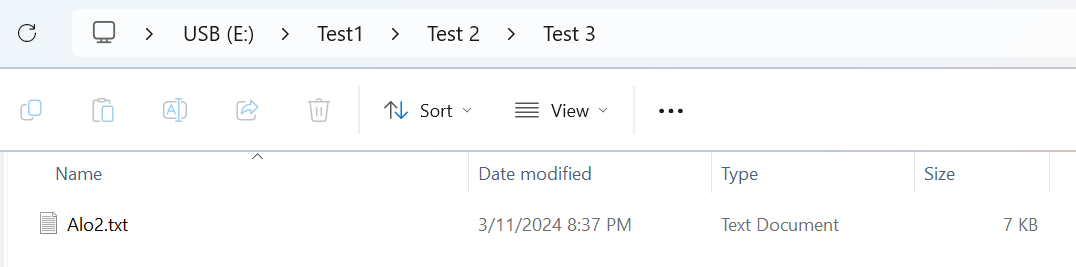
* Trong thư mục Test1 sẽ có 1 tập tin văn bản và 1 thư mục khác



* Trong thư mục Test2 sẽ có thêm 1 thư mục nữa

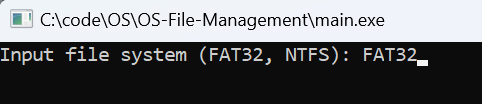


* Và trong thư mục Test3 sẽ là 1 tập tin văn bản



### Sử dụng chương trình để đọc ổ đĩa

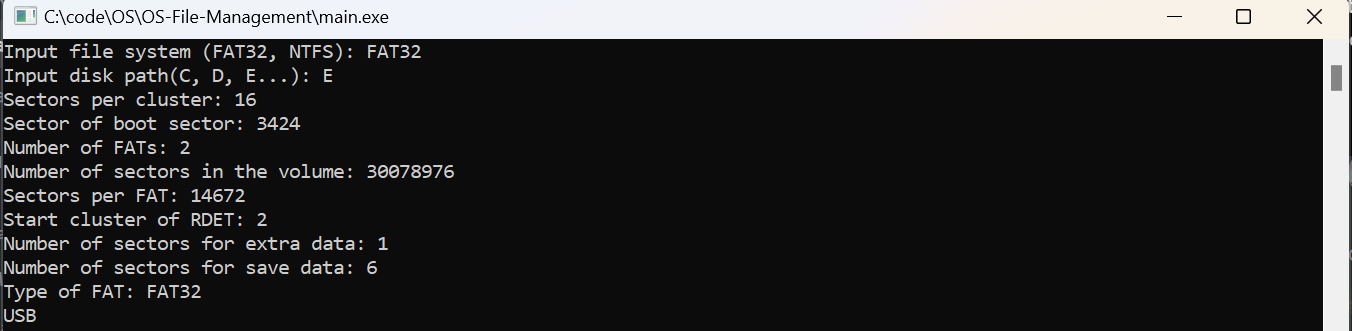
* Chạy tập tin thực thi main.exe, màn hình sẽ hiện ra, yêu cầu chọn loại hệ thống quản lý tập tin



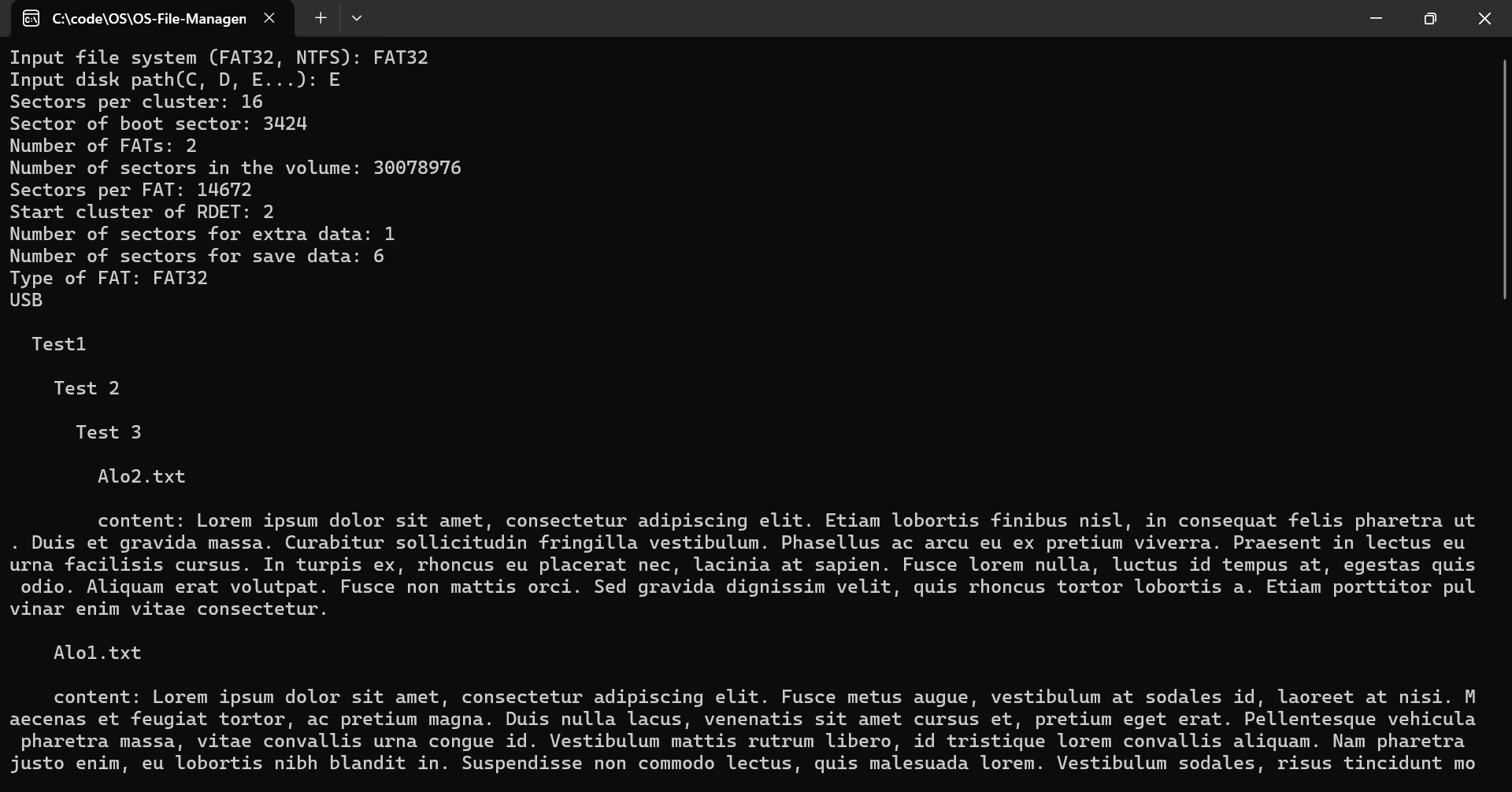
* Chương trình sẽ tiếp tục yêu cầu chọn ổ đĩa để đọc



* Sau khi đã điền chính xác cả 2 mục, toàn bộ nội dung của ổ đĩa sẽ được in ra, bao gồm một số thông tin về hệ thống quản lý

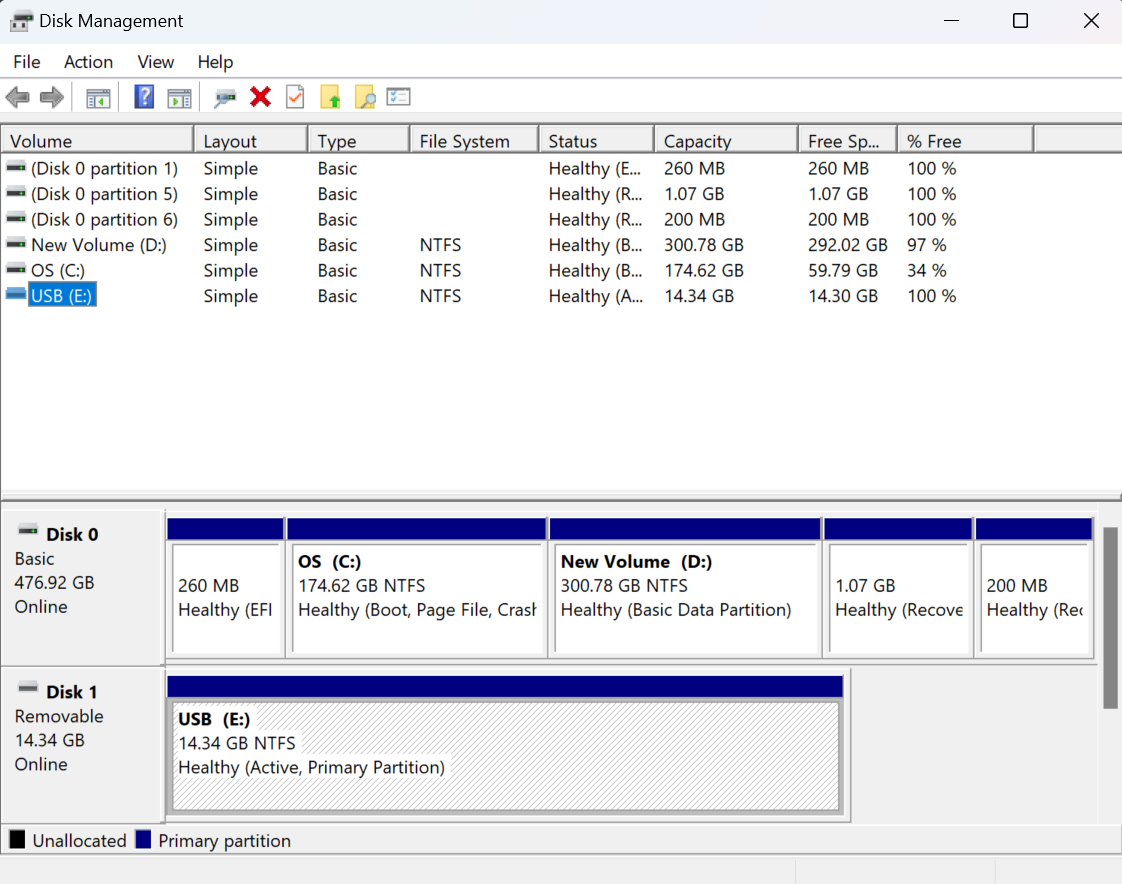


và cây thư mục. Trong đó, nội dung của các tập tin văn bản cũng sẽ được in ra



* Nhấn phím bất kỳ để tắt chương trình.

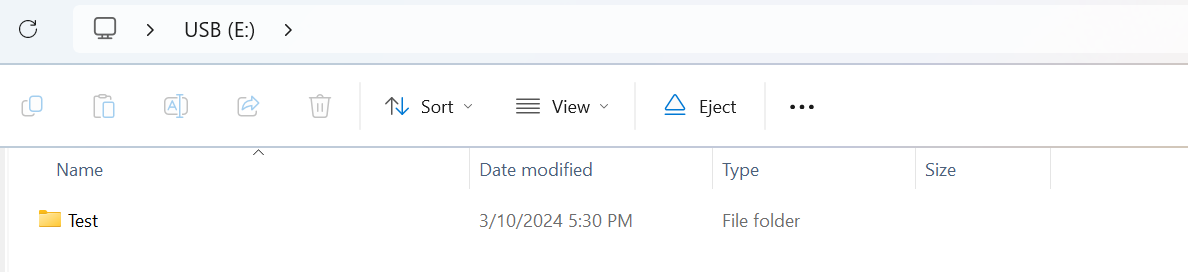
## NTFS



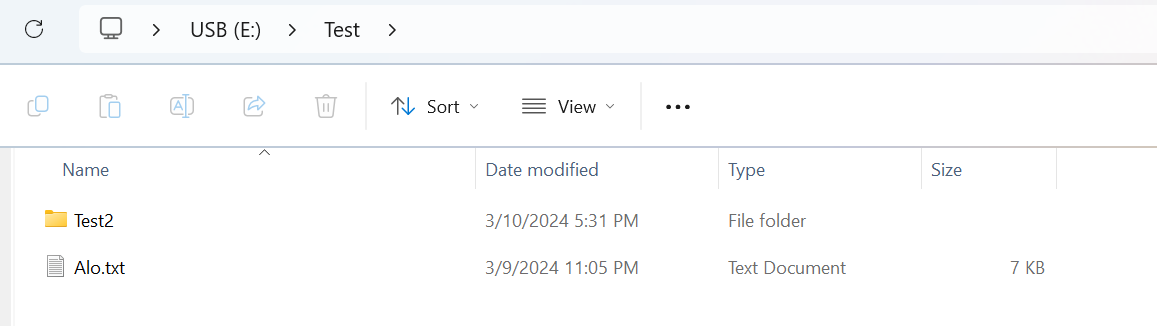
Ổ đĩa được chọn để demo lần này cũng sẽ là ổ đĩa E

### Nội dung ổ đĩa

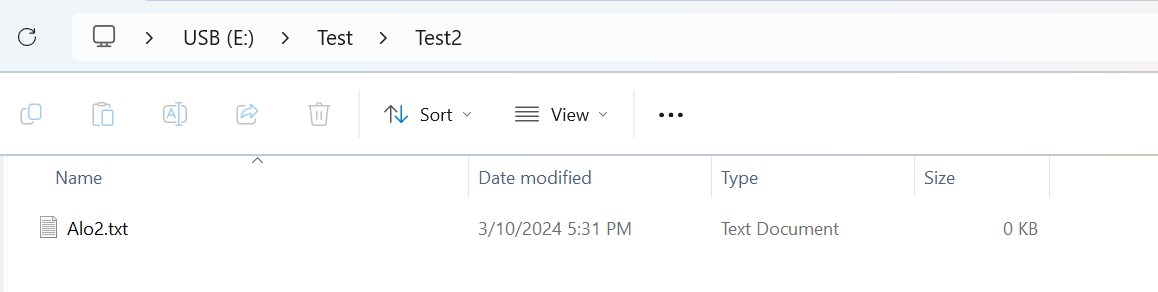
* Ổ đĩa E bao gồm 1 thư mục



* Trong thư mục Test có 1 thư mục khác, và 1 tập tin văn bản

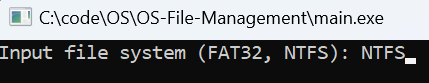


* Và trong thư mục Test2 có duy nhất 1 tập tin văn bản khác

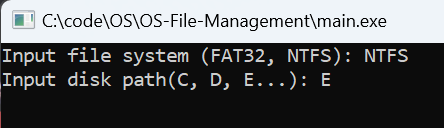


### Sử dụng chương trình để đọc ổ đĩa

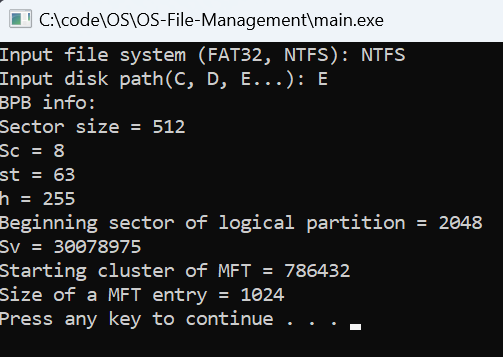
* Chạy tập tin thực thi main.exe, màn hình sẽ hiện ra, yêu cầu chọn loại hệ thống quản lý tập tin



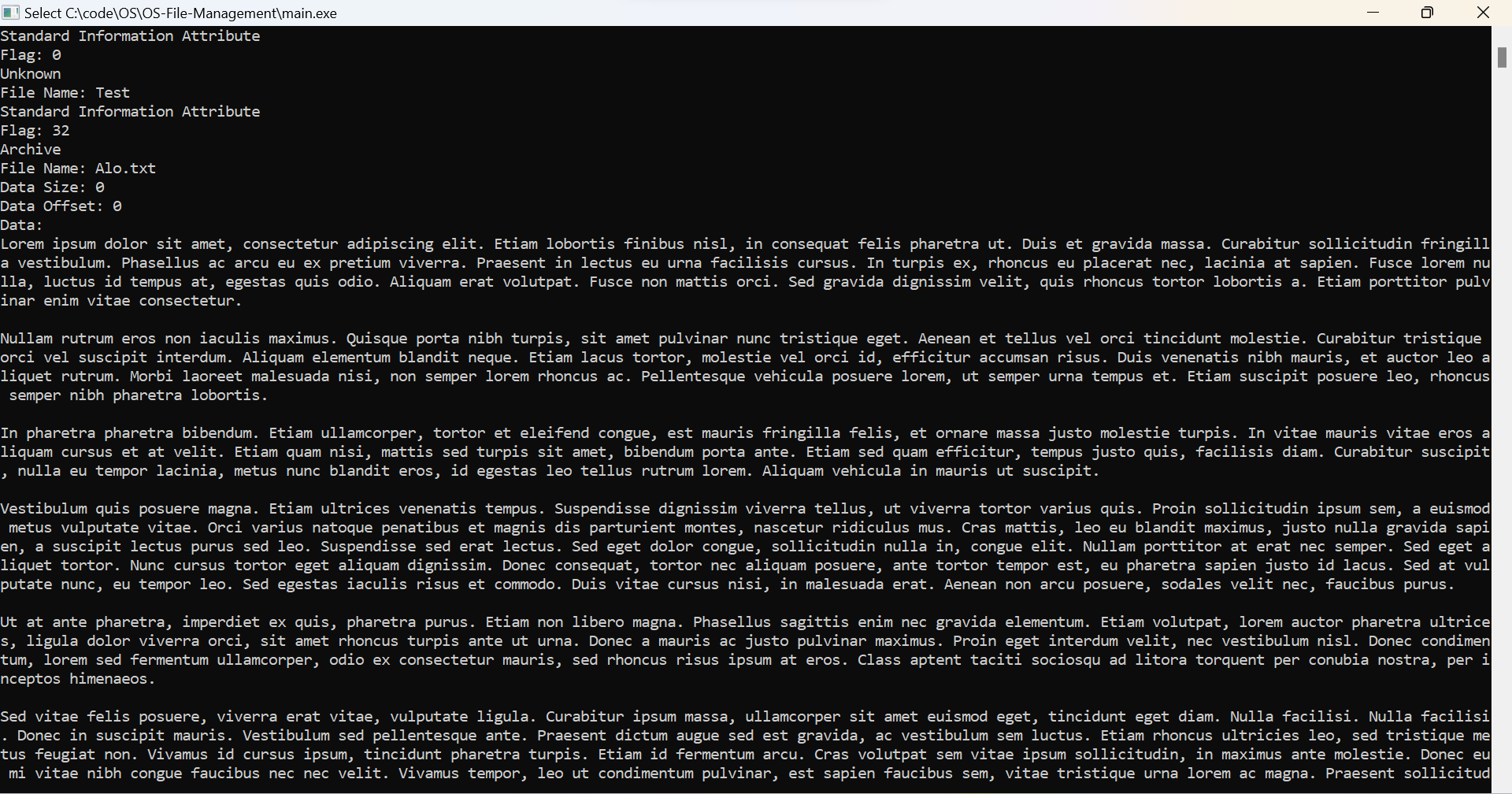
* Chương trình sẽ tiếp tục yêu cầu chọn ổ đĩa để đọc



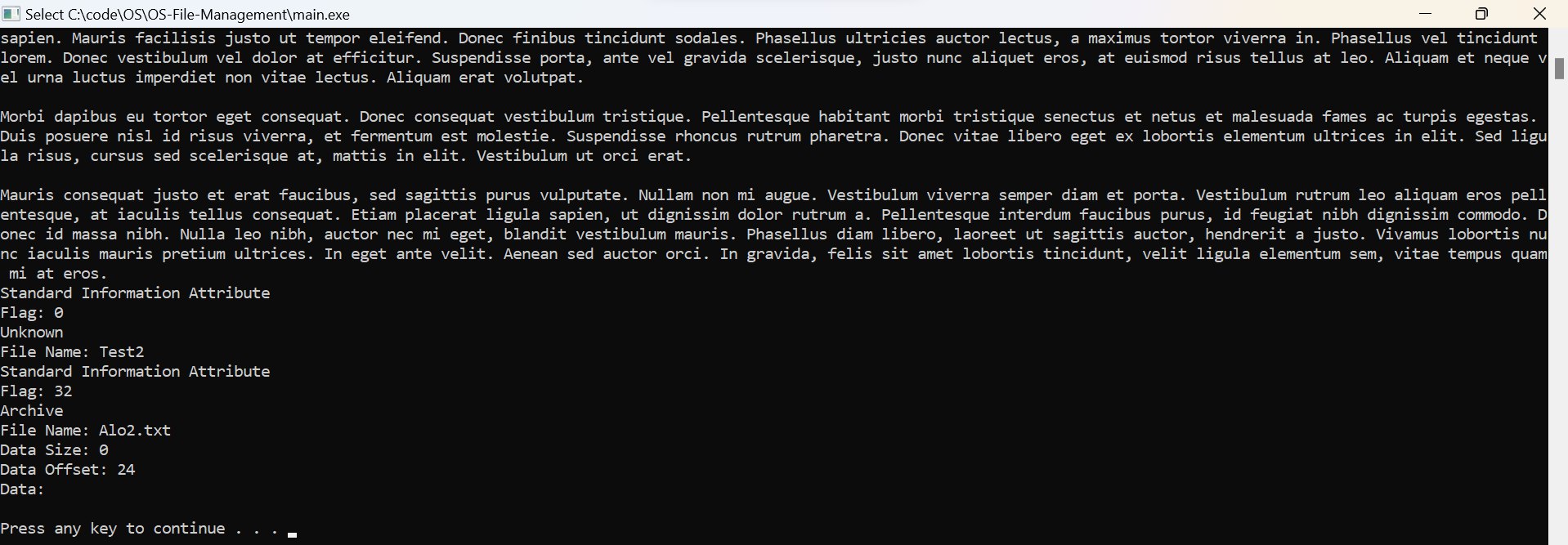
* Sau khi đã điền chính xác cả 2 mục, thông tin của hệ thống quản lý sẽ được in ra. Chương trình sẽ tạm ngưng để người dùng có thể đọc thông tin



* Nhấn phím bất kỳ để tiếp tục chương trình, thông tin của tất cả các attribute của tất cả các entry sẽ được in ra.
* Trong demo này, vì attribute $DATA của file Alo.txt rất lớn nên phải lưu dưới dạng non-resident, vì vậy giá trị Data size và Data offset đều bằng 0



* Tập tin văn bản Alo2.txt thì có attribute $DATA dạng resident, nhưng không có nội dung nên giá trị Data size vẫn bằng 0, nhưng giá trị Data offset thì khác 0



* Nhấn phím bất kỳ để kết thúc chương trình.

# Đóng góp

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Thành viên | MSSV | Công việc | Tỉ lệ (%) hoàn thành | Tổng đóng góp cho nhóm |
| Trần Nguyễn Minh Hoàng | 22127131 |  | 100% | 20% |
| Lê Hồ Phi Hoàng | 22127123 |  | 100% | 20% |
| Đỗ Đình Hải | 22127095 |  | 100% | 20% |
| Phạm Thành Đạt | 22127064 |  | 100% | 20% |
| Đỗ Phan Tuấn Đạt | 22127057 |  | 100% | 20% |

# Tài liệu tham khảo

* Slide tài liệu Hệ điều hành của thây Lê Viết Long, ĐHKHTN-ĐHQG HCM
* Source code Read Sector C++ cung cấp bởi thầy Lê Viết Long
* <https://legiacong.blogspot.com/2014/05/he-thong-quan-ly-tap-tin-ntfs-8.html>
* <http://inform.pucp.edu.pe/~inf232/Ntfs/ntfs_doc_v0.5/attributes/attribute_list.html>
* <https://www.kcall.co.uk/ntfs/index.html?fbclid=IwAR0_ve_jzXeksnhJD3_q7JAAr6PuazBDYy277tIfBtB-LX53UwiMADvLcWU>
* <https://www.youtube.com/watch?v=6WFUM5eViIk>