# Class entry

Là một lớp đại diện cho một entry trong hệ thống tệp tin.

## Class entry này gồm các thuộc tính như sau

* **name**: Một chuỗi (string) lưu trữ tên của một file/folder.
* **status**: Một biến byte (BYTE) lưu trữ trạng thái của entry đó thuộc loại trống, bị xóa hay đang lưu thông tin của file khác.
* **size**: Một số nguyên (int) lưu trữ kích thước của phần nội dung tập tin.
* **firstCluster**: Một số nguyên (int) lưu trữ số cluster đầu tiên của entry.
* **attribute**: Thuộc tính của entry, được biểu diễn bằng một enum **EntryAttribute**, mô tả các thuộc tính như Read-only, Hidden, System, Volume Label, Directory, Archive.
* **entries**: Một vector chứa các vector chuỗi, dùng để lưu trữ các thông tin về entry phụ và entry chính với dạng từng string là 1 byte hex.

## Class entry bao gồm các chức năng sau:

### Constructor:

* **name = "";** Gán giá trị rỗng cho thuộc tính **name**
* **status = 0;** Gán giá trị 0 cho thuộc tính **status**, tức là không có trạng thái nào được thiết lập ban đầu cho mục.
* **size = 0;** Gán giá trị 0 cho thuộc tính **size**, tức là kích thước của mục ban đầu là 0.
* **firstCluster = 0;** Gán giá trị 0 cho thuộc tính **firstCluster**,
* **attribute = (EntryAttribute)0;**: Gán giá trị 0 cho thuộc tính **attribute**.

Điều này đảm bảo rằng mỗi khi một đối tượng **Entry** được tạo mới và các thuộc tính của nó sẽ trống hoàn toàn

### operator=

Hàm này thực hiện việc gán giá trị của một đối tượng Entry (entry) cho một đối tượng Entry khác (\*this).



Đoạn mã trên sao chép toàn bộ thuộc tính bao gồm (name, status, size, firstCluster, attribute, entries) từ đối tượng entry sang entry hiện tại (this), Cuối cùng, phương thức trả về một tham chiếu đến đối tượng hiện tại bằng cách sử dụng return \*this**.** Cho phép chuỗi gán liên tiếp và sử dụng dấu '=' cho các đối tượng của lớp Entry.

### readEntry

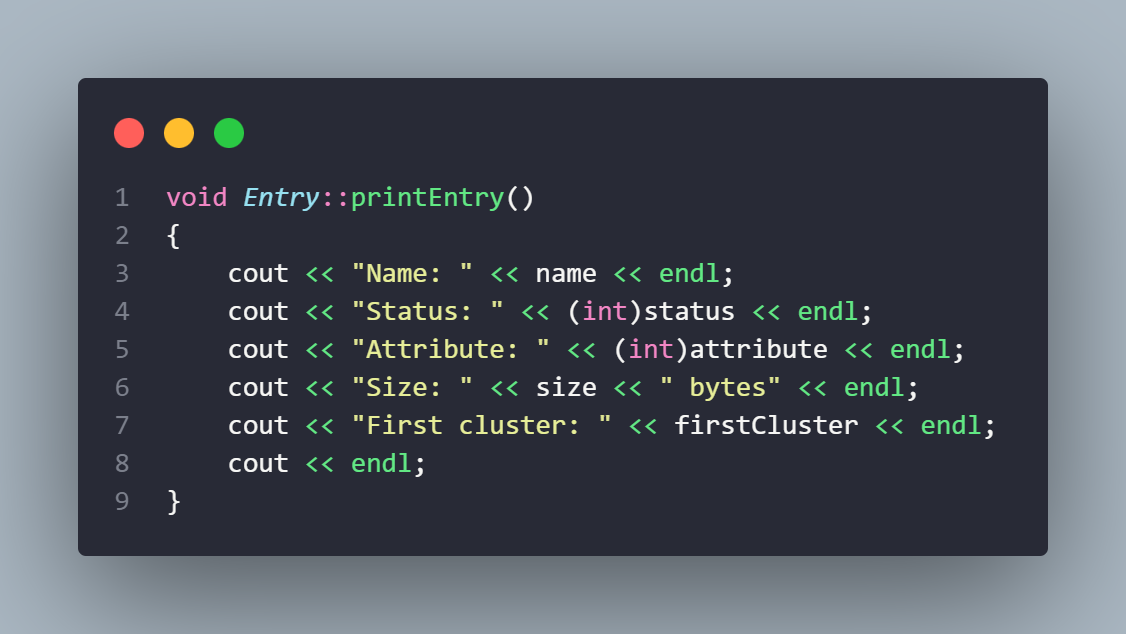
Được viết để đọc dữ liệu từ một vector<vector<string>> với mỗi vector<string> đại diện cho một entry



* + this->entries = entry lưu lại dữ liệu từ entry vào entries của đối tượng.
  + vector<string> primaryEntry = this->entries[this->entries.size() - 1] :Lấy ra entry chính lưu vào primaryEntry để bắt đầu việc đọc entry.
  + this->status = (convertHexadecimalToDecimal(primaryEntry[0])); Chuyển đổi giá trị Hexadecimal ở vị trí đầu tiên của `primaryEntry` thành Decimal và gán cho `status` của đối tượng `Entry`.
  + if (this->entries.size() > 1)`: Kiểm tra nếu có nhiều hơn một mục trong `entries`.
    - Trong trường hợp có nhiều hơn một entry trong danh sách các entry:
      * Sẽ gọi hàm getFullNameFromASetOfEntry để lấy tên đầy đủ của mục và gán cho thuộc tính name của đối tượng `Entry`.
    - Trong trường hợp chỉ có một mình entry chính:
      * Lặp qua các ký tự Hexadecimal trong `primaryEntry` để tạo tên của mục. Các ký tự không phải là "20" hoặc "00" được thêm vào tên.
      * Chuyển đổi tên từ mã Hexadecimal sang UTF-16.
      * Gán tên đã được chuyển đổi cho thuộc tính `name`.
  + this->attribute = (EntryAttribute)convertHexadecimalToDecimal(primaryEntry[0xB]); Chuyển đổi giá trị Hexadecimal ở vị trí 0xB của entry chính thành Decimal và gán cho thuộc tính attribute của đối tượng Entry.
  + temp = convertStringToLittleEdian(getStringFromVector(primaryEntry, 0x1C, 4)) + this->size = convertHexadecimalToDecimal(temp); Lấy ra một chuỗi từ entry chính bắt đầu từ vị trí 0x1C với độ dài 4 và chuyển đổi sang Little Endian. Sau đó lại chuyển đổi chuỗi đã chuyển đổi sang Decimal và gán cho thuộc tính `size` của đối tượng `Entry`.
  + temp = convertStringToLittleEdian(getStringFromVector(primaryEntry, 0x14, 2)) + convertStringToLittleEdian(getStringFromVector(primaryEntry, 0x1A, 2));`: Lấy ra hai chuỗi từ `primaryEntry`, mỗi chuỗi bắt đầu từ các vị trí 0x14 và 0x1A, mỗi chuỗi có độ dài 2, và chuyển đổi sang Little Endian.
  + this->firstCluster = convertHexadecimalToDecimal(temp) Chuyển đổi chuỗi đã chuyển đổi sang Decimal và gán cho thuộc tính `firstCluster` của đối tượng `Entry.

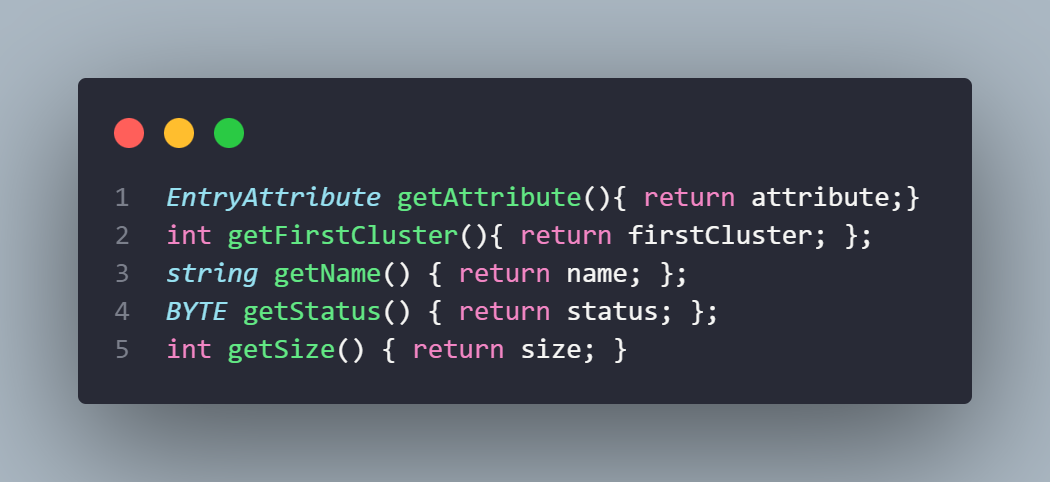
### printEntry

Hàm này in ra thông tin của một entry trong hệ thống tập tin ra màn hình:



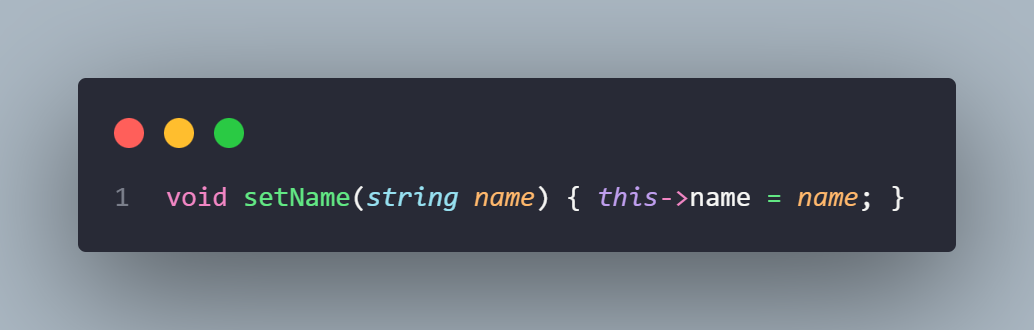
Thông tin được in ra bao gồm : **name, status, attribute, size, firstCluster** của entry.

### Các hàm get



* **getAttribute()**: Phương thức này trả về giá trị của thuộc tính **attribute**, là một đối tượng thuộc kiểu **EntryAttribute**. Nó sẽ trả về một enum **EntryAttribute** biểu diễn các thuộc tính của mục.
* **getFirstCluster()**: Phương thức này trả về giá trị của thuộc tính **firstCluster**, đại diện cho số cluster đầu tiên của mục.
* **getName()**: Phương thức này trả về giá trị của thuộc tính **name**, là tên của mục.
* **getStatus()**: Phương thức này trả về giá trị của thuộc tính **status**, đại diện cho trạng thái của mục.
* **getSize()**: Phương thức này trả về giá trị của thuộc tính **size**, là kích thước của mục

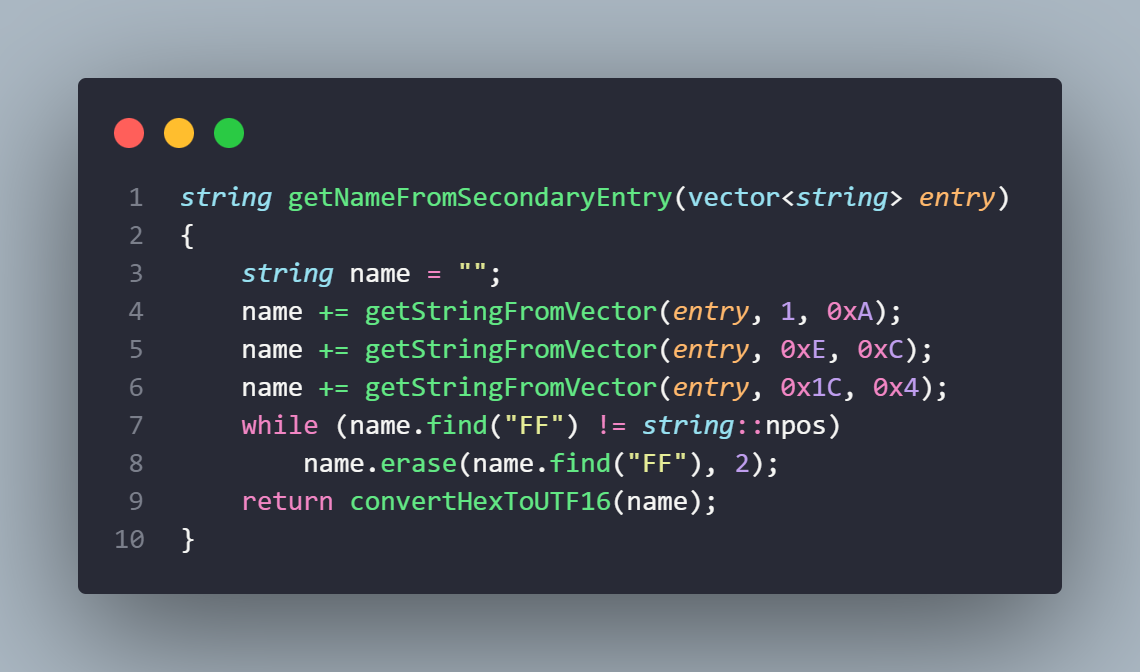
### setName



Hàm này dùng để đặt tên cho entry.

### Hàm **getNameFromSecondaryEntry** và **getFullNameFromASetOfEntry**

* Hàm **getNameFromSecondaryEntry**



- Hàm này nhận một vector chuỗi entry đại diện cho một mục phụ và trích xuất tên của mục từ các phần của vector entry.

- Cụ thể, hàm này sử dụng hàm getStringFromVector để trích xuất các phần của vector entry ứng với các vị trí cụ thể và kích thước cụ thể của chuỗi.

- Sau đó, hàm loại bỏ mọi cặp ký tự "FF" khỏi tên bằng cách lặp qua chuỗi và xóa mỗi cặp "FF" một cách lặp đi lặp lại.

- Cuối cùng, tên được chuyển từ mã Hexadecimal sang UTF-16 bằng cách sử dụng hàm **convertHexToUTF16**, và tên kết quả được trả về.

* Hàm **getFullNameFromASetOfEntry**



- Hàm này nhận một vector các vector chuỗi entry, mỗi vector chuỗi đại diện cho một entry.

- Hàm này lặp qua các vector chuỗi trong entry, trừ vector cuối cùng (vector chuỗi cuối cùng thường chứa dữ liệu của entry chính).

- Với mỗi vector chuỗi (trừ vector cuối cùng), hàm gọi hàm getNameFromSecondaryEntry để trích xuất tên của entry phụ và nối vào biến name.

- Kết quả là tên đầy đủ của entry được xây dựng từ các tên của các entry phụ, và được trả về.

Cả hai hàm này đều được sử dụng để xử lý dữ liệu mục từ các vector chuỗi, trích xuất tên của mục và chuẩn hóa nó trước khi trả về.

### Hàm getFile



1. **Khởi tạo đối tượng File mới**:
   * Hàm bắt đầu bằng việc tạo một con trỏ đến một đối tượng **File** mới thông qua lệnh **File\* a = new File;**.
2. **Đọc dữ liệu từ các sector**:
   * Sử dụng một vòng lặp để lặp qua các phần tử trong **list\_Sector**.
   * Đối với mỗi sector được chỉ định, hàm đọc dữ liệu từ sector đó vào một mảng byte **data** bằng cách sử dụng hàm **readSector**.
   * Dữ liệu từ sector đó được thêm vào vector **binary\_data**.
3. **Chuyển đổi dữ liệu thành chuỗi Hexadecimal**:
   * Sau khi đã đọc xong dữ liệu từ các sector, hàm tiến hành chuyển đổi dữ liệu từ vector **binary\_data** thành một chuỗi Hexadecimal.
   * Dữ liệu Hexadecimal này được lưu vào biến **content**.
4. **Đặt dữ liệu vào đối tượng File và trả về**:
   * Dữ liệu Hexadecimal được chuyển đổi sang chuỗi UTF-16 bằng cách sử dụng hàm **convertHexToUTF16**.
   * Chuỗi UTF-16 này được đặt vào đối tượng **File** thông qua phương thức **setData**.
   * Cuối cùng, đối tượng **File** được trả về.

# Class Item

Đây là một lớp cơ sở cho class File và class Folder kế thừa.

## Class Item có một thuộc tính

**Entry \*entry**: Một con trỏ đến một đối tượng **Entry**. Điều này cho phép mỗi **Item** có một liên kết với một mục trong hệ thống tập tin

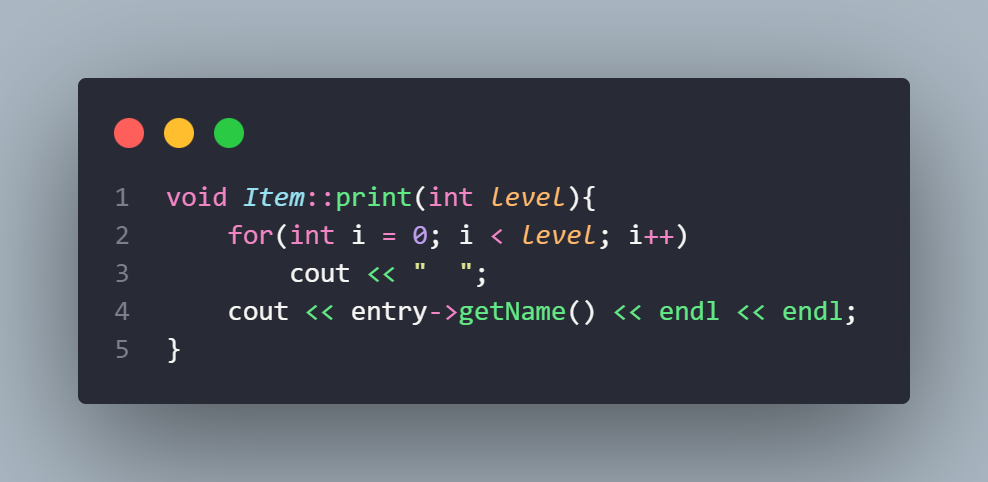
## Class Item có các chức năng:

### setEntry



Hàm này dùng để kết nối Item với một đối tượng **Entry**, giúp trong việc lưu trữ thông tin và thực hiện các thao tác liên quan đến mục trong hệ thống tập tin.

### Print

  
Cùng với một số dấu cách để tạo ra hiệu ứng thụt lề dựa trên cấp độ (level) được chỉ định. Sau đó sẽ in ra tên của Item đó.

### deleteItem



Xóa vùng nhớ mà con trỏ entry đang chỉ đến.

# Class File

## Thuộc tính:

Class File có 1 thuộc tính là **data** dùng để lưu nội dung của file đó. Ngoài ra File còn kế thừa thuộc tính của Item là Entry.

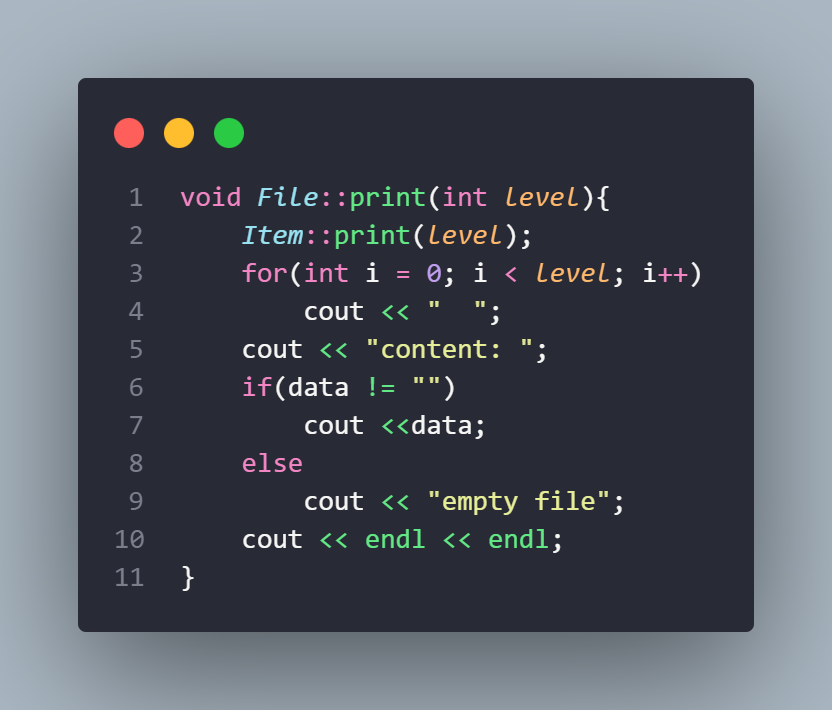
## Các chức năng

### setData



Sau khi đọc được nội dung của file thì sẽ gọi hàm setData để lưu nội dung vào thuộc tính data của File.

### Print



* **Tham số**: Một tham số **level** có giá trị mặc định là 0, đại diện cho cấp độ thụt lề.
* **Hoạt động**:
  + Đầu tiên, phương thức gọi phương thức **print** của lớp cơ sở **Item** để in ra tên của tệp với cấp độ thụt lề đã được chỉ định.
  + Tiếp theo, vòng lặp được sử dụng để in ra số dấu cách tương ứng với cấp độ thụt lề lần nữa, tạo ra hiệu ứng thụt lề khi in ra nội dung của tệp.
  + Sau đó, dòng "content: " được in ra để chỉ định phần nội dung của tệp.
  + Tiếp theo, kiểm tra xem dữ liệu của tệp có trống không hay không. Nếu không trống, nó sẽ in ra dữ liệu của tệp. Nếu trống, nó sẽ in ra "empty file".
  + Cuối cùng, một dòng trống được in ra để phân tách giữa các mục.

# Class Folder

## Thuộc tính:

Class Folder có 1 thuộc tính là **vector<Item\*> subfolder** dùng để lưu file hoặc subfolder bên trong folder hiện tại.

## Các chức năng

### addItem

Hàm này dùng để thêm file hoặc subfolder vào subfolder bằng cách gọi hàm push\_back của vector để thêm Item\* vào subfolder.



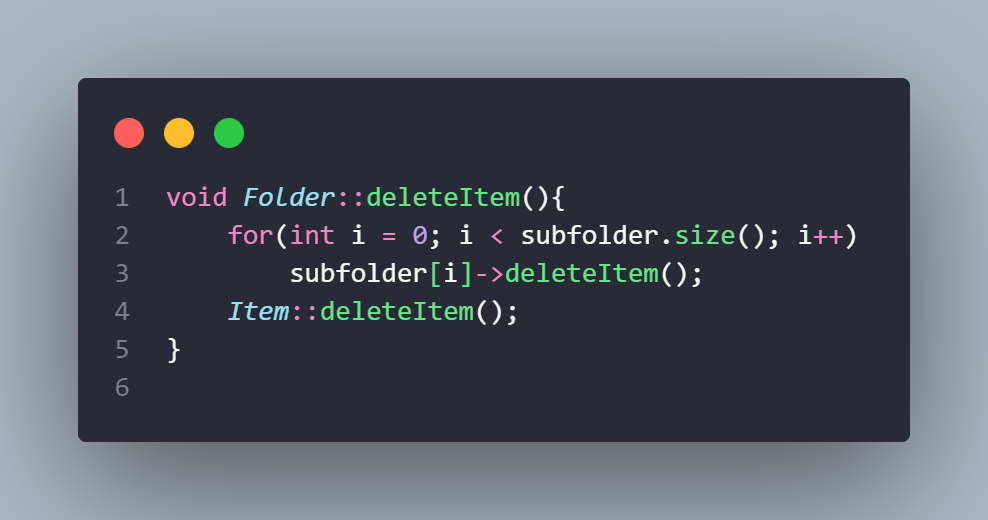
### Print

A screen shot of a computer program

Description automatically generated

* Đầu tiên, phương thức gọi phương thức **print** của lớp cơ sở **Item** để in ra tên của thư mục với cấp độ thụt lề đã được chỉ định.
* Tiếp theo, một vòng lặp được sử dụng để lặp qua tất cả các mục con trong vector **subfolder**.
* Trong mỗi lần lặp, phương thức **print** của mỗi mục con được gọi, với cấp độ thụt lề tăng lên 1 so với cấp độ hiện tại. Điều này đảm bảo rằng mỗi mục con được in ra với một cấp độ thụt lề cao hơn so với thư mục cha.
* Qua việc gọi đệ quy, tất cả các mục con được in ra đúng với cấp độ thụt lề phù hợp.

### deleteItem



* + Đầu tiên, một vòng lặp được sử dụng để lặp qua tất cả các mục con trong vector **subfolder**.
  + Trong mỗi lần lặp, phương thức **deleteItem** của mục con được gọi, đảm bảo rằng tất cả các mục con cũng được xóa.
  + Sau khi đã xóa tất cả các mục con, phương thức **deleteItem** của lớp cơ sở **Item** được gọi để xóa chính thư mục đó.

Hàm này đảm bảo rằng khi một thư mục bị xóa, tất cả các mục con của nó cũng sẽ được xóa đi.

# Class Entries

Lớp **Entries** được thiết kế để quản lý một danh sách các **Entry**, cũng như cung cấp các phương thức để thực hiện các thao tác trên danh sách này. Dưới đây là mô tả của các thành phần và phương thức trong lớp **Entries**:

## Thuộc tính:

* **vector<Entry\*> entries**: Một vector chứa các con trỏ đến các đối tượng **Entry**.

## Chức năng

### Input

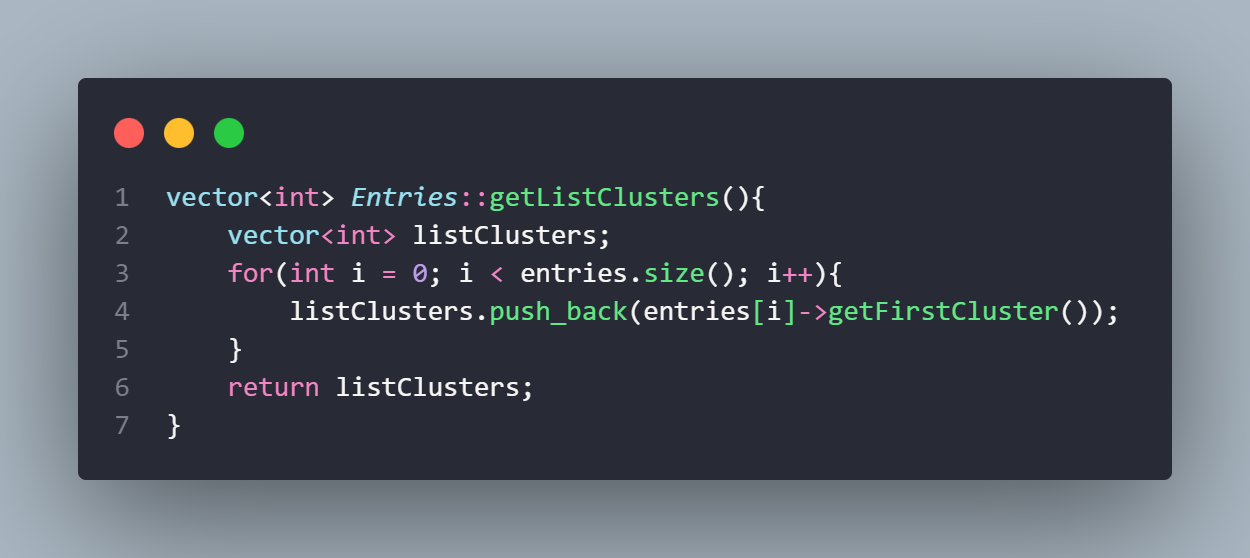
Hàm này cho phép nhập danh sách các mục từ dữ liệu đầu vào và lưu trữ chúng trong danh sách của đối tượng **Entries**, loại bỏ các mục có thuộc tính không mong muốn.



* Trong vòng lặp **while**, phương thức sẽ tiếp tục thực hiện cho đến khi danh sách **entries** trống.
* Trong mỗi vòng lặp, phương thức **extractEntry** được gọi để rút trích một mục từ danh sách **entries**. Kết quả được lưu vào biến **temp**, là một vector hai chiều chứa thông tin về một mục.
* Một đối tượng mới của lớp **Entry** được tạo ra bằng cách sử dụng toán tử **new**.
* Phương thức **readEntry** của đối tượng **Entry** mới được gọi để đọc thông tin từ vector **temp** và cập nhật các thuộc tính của mục.
* Kiểm tra thuộc tính của mục (**getAttribute**). Nếu thuộc tính khác **0x16**, tức là mục không phải là mục được xác định trước (predefined), đối tượng **Entry** được thêm vào danh sách **this->entries**.
* Nếu mục có thuộc tính là **0x16**, đối tượng **Entry** được xóa để tránh rò rỉ bộ nhớ

### getListClusters

Hàm này sẽ trả về danh sách các cluster bắt đầu của mỗi entry trong vector entries



* Tạo một vector **listClusters** để lưu trữ danh sách các cluster.
* Trong vòng lặp **for**, phương thức duyệt qua từng mục trong danh sách **entries**.
* Đối với mỗi mục, phương thức gọi phương thức **getFirstCluster** để lấy thông tin về cluster đầu tiên của mục đó.
* Giá trị cluster được thu được từ mỗi mục được thêm vào vector **listClusters**.
* Khi đã duyệt qua tất cả các mục, vector **listClusters** chứa danh sách các cluster của các mục và được trả về.

### getListSector



* 1. Sử dụng phương thức **getListClusters** để thu thập danh sách các cluster được sử dụng bởi các mục trong danh sách **entries**.
  2. Sử dụng bảng Fat (**fatTable**) để tạo danh sách các cluster của từng mục bằng cách gọi phương thức **listClustersOfEntry** với danh sách các cluster đã thu thập ở bước trước.
  3. Duyệt qua danh sách các cluster của từng mục và tính toán offset của mỗi cluster dựa trên thông tin từ bảng Boot Sector và bảng Fat:
     + Đối với mỗi mục, một vector **listSector** được tạo ra để lưu trữ thông tin về các sector được sử dụng bởi mục đó.
     + Đối với mỗi cluster của mục, tính toán offset của cluster bằng cách sử dụng công thức:

**offset = SB + SF \* NF + (cluster - 2) \* SC**

* + - trong đó:
      * **SB**: Số sector trước phân vùng FAT.
      * **SF**: Số sector cho mỗi phân vùng FAT.
      * **NF**: Số phân vùng FAT.
      * **SC**: Số sector cho mỗi cluster.
      * **cluster**: Số thứ tự của cluster.
    - Offset của mỗi cluster được thêm vào vector **listSector**.
  1. Sau khi đã tính toán offset cho tất cả các cluster của mỗi mục, vector **listSector** của mỗi mục được thêm vào vector **list\_Sector\_Of\_Entry**.
  2. Khi đã duyệt qua tất cả các mục trong danh sách **entries**, vector **list\_Sector\_Of\_Entry** chứa thông tin về các sector được sử dụng bởi các mục và được trả về.

### getRootDirectory

Hàm này tạo một đối tượng thư mục (**Folder**) để đại diện cho thư mục gốc của hệ thống tập tin, và điều hướng qua danh sách các mục trong **entries** để tạo các mục con trong thư mục gốc.



* Sử dụng phương thức **getListSector** để lấy danh sách các sector được sử dụng bởi các mục trong **entries**.
* Tạo một đối tượng thư mục (**Folder**) mới để đại diện cho thư mục gốc và gán một entry của **entries** đầu tiên vào thư mục gốc này.
* Duyệt qua các mục trong danh sách **entries**, bắt đầu từ vị trí thứ hai (vì entry đầu tiên đã được gán cho thư mục gốc):
  + Nếu mục là thư mục (có thuộc tính là **Directory**), tiến hành đọc tất cả các mục con trong thư mục này bằng cách gọi hàm **readEntireEntries**, sau đó tạo một đối tượng mới của lớp **Entries** và gọi phương thức **input** để nhập các mục con vào đối tượng này. Tiếp theo, gọi hàm **getSubDirectory** để tạo một đối tượng thư mục con cho thư mục này.
  + Nếu mục là một tập tin (có thuộc tính là **Archive**), sử dụng hàm **getFile** để tạo một đối tượng tập tin (**File**) cho mục này.
  + Sau khi đã tạo đối tượng thích hợp, thêm đối tượng này vào thư mục gốc.
* Trả về đối tượng thư mục gốc đã được tạo.

getSubDirectory



* Xóa mục thứ hai (vị trí 1) trong danh sách **entries** bằng cách sử dụng hàm **erase**, đây là entry miêu tả cha của entry này.
* Thiết lập tên mới cho mục đầu tiên (vị trí 0) trong danh sách **entries** bằng cách gọi phương thức **setName** với đối số **name**.
* Gọi lại phương thức **getRootDirectory** để tạo lại thư mục gốc với tên mới và trả về thư mục này.

# Hàm linh tinh

## readEntireEntries

Hàm `readEntireEntries` có một số chức năng chính:



* Đọc toàn bộ các entry từ một ổ đĩa:

- Hàm này nhận đường dẫn đến ổ đĩa (`diskPath`), vị trí sector bắt đầu của Region Directory Entry Table (RDET) (`startSectorOfRDET`), và một vector tham chiếu `entries` để lưu trữ các mục được đọc.

Mục đích của hàm là đọc toàn bộ các mục từ ổ đĩa và lưu chúng vào vector `entries`.

* Đọc dữ liệu từ sector:

- Sử dụng hàm **readSector** để đọc dữ liệu từ sector của ổ đĩa và lưu vào vector **storedValues**. Hàm **readSector** được gọi mỗi lần vòng lặp chạy.

- Hàm readSector nhận vào đường dẫn của ổ đĩa, vị trí sector cần đọc, kích thước sector và một vector tham chiếu để lưu trữ dữ liệu đọc được.

* Kiểm tra điều kiện dừng vòng lặp

- Sau mỗi lần đọc sector, hàm kiểm tra xem dữ liệu đọc được có trống không (**checkEmpty**) hoặc vector storedValues có trống không hay không (**storedValues.empty()**). Nếu có, vòng lặp sẽ dừng lại.

* Xử lý mỗi entry:

- Vòng lặp **for** chạy qua mỗi phần tử của vector storedValues với bước nhảy là 32, vì mỗi mục thường có độ dài 32 bytes.

- Mỗi lần lặp, tạo một vector entry để lưu trữ dữ liệu. Sau đó, các phần tử trong storedValues từ vị trí hiện tại đến vị trí hiện tại + 31 được thêm vào `entry`.

- Nếu entry được đọc trống (có thể là kết thúc hoặc bị xóa), vòng lặp sẽ dừng lại bằng lệnh **break**.

* Lưu trữ entry vừa đọc được:

- Sau khi một mục được đọc và lưu trữ trong `entry`, vector `entry` này được thêm vào vector `entries`, đại diện cho toàn bộ các mục được đọc.

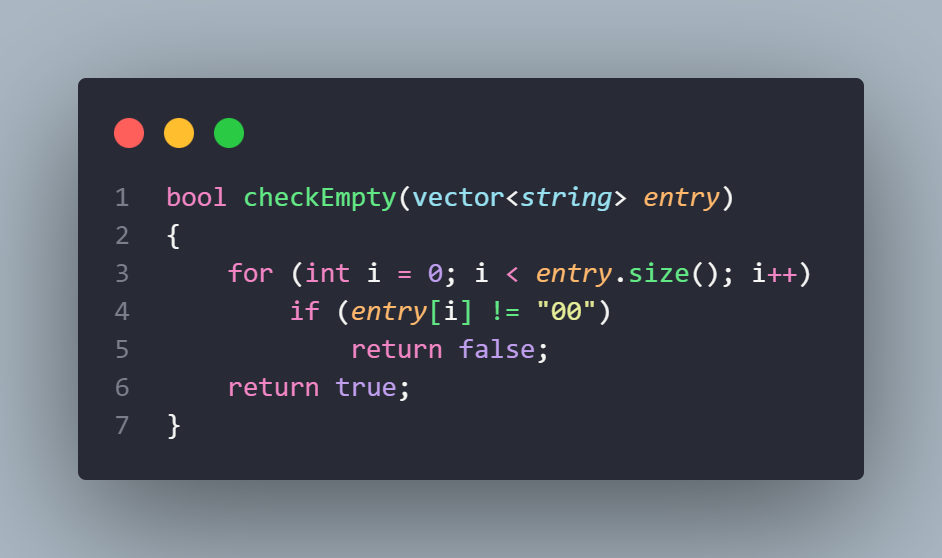
Tăng vị trí sector và làm sạch dữ liệu

- Sau khi đọc một sector, vị trí sector được tăng lên để đọc sector tiếp theo.

- Dữ liệu trong vector `storedValues` được xóa trước khi tiếp tục vòng lặp để lưu trữ dữ liệu mới từ sector tiếp theo.

## checkEmpty

Hàm **checkEmpty** được thiết kế để kiểm tra xem một vector **entry** có rỗng hay không.



* Trong vòng lặp **for**, hàm duyệt qua từng phần tử trong vector **entry**.
* Trong mỗi vòng lặp, hàm kiểm tra xem giá trị của phần tử đó có phải là **"00"** hay không.
* Nếu tìm thấy một phần tử có giá trị khác **"00"**, hàm trả về **false** ngay lập tức, báo hiệu rằng vector không rỗng.
* Nếu không có phần tử nào có giá trị khác **"00"** sau khi duyệt qua toàn bộ vector, hàm trả về **true**, cho biết vector là rỗng.

## checkPrimary

A screen shot of a computer code

Description automatically generated

Hàm này mục đích để kiểm tra xem entry này có phải entry chính hay không. Bằng cách:

* Hàm kiểm tra giá trị của phần tử thứ 11 trong vector **entry** (vị trí 10 khi tính từ 0).
* Nếu giá trị của phần tử này là **"0F"**, hàm trả về **false** ngay lập tức, báo hiệu rằng mục không phải là mục chính.
* Nếu giá trị của phần tử không phải là **"0F"**, hàm trả về **true**, cho biết mục là một mục chính.

## extractEntry



* Tạo một vector **result** để lưu trữ danh sách các mục đã được rút trích.
* Duyệt qua từng mục trong danh sách **entries**:
  + Thêm mục hiện tại vào vector **result**.
  + Kiểm tra xem mục hiện tại có thỏa mãn điều kiện được xác định bởi hàm **checkPrimary** hay không.
    - Nếu có, hàm sẽ loại bỏ tất cả các mục từ vị trí hiện tại đến mục thỏa mãn điều kiện đó và kết thúc vòng lặp.
    - Nếu không, hàm sẽ tiếp tục lặp qua các mục khác.
* Trả về vector **result** chứa danh sách các mục đã được rút trích.

## removeFaultyEntry

Hàm này dùng để xóa các entry mang trạng thái E5 ra khỏi danh sách các entries bởi vì đây là các entry đã bị xóa.

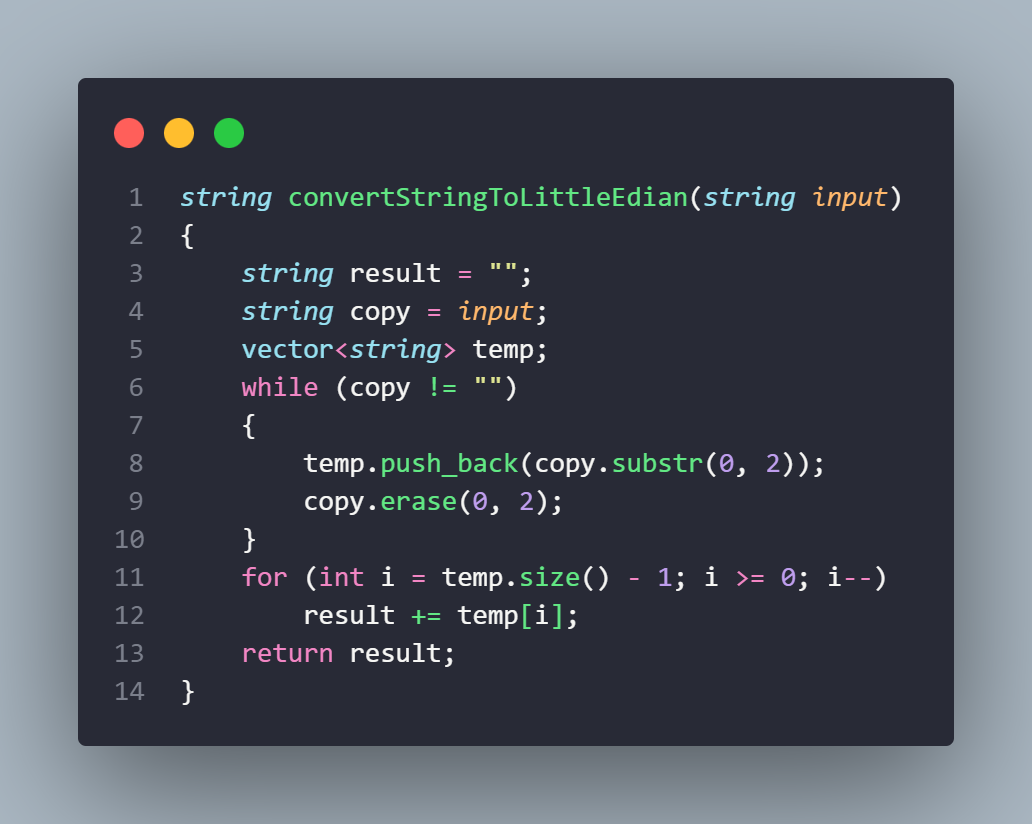


Cách hoạt động:

* Duyệt qua từng mục trong danh sách **entries**.
* Kiểm tra xem mục hiện tại có phải là mục không hợp lệ hay không. Trong trường hợp này, mục không hợp lệ được xác định bởi giá trị của phần tử đầu tiên trong mục (vị trí 0) là **"E5"**.
* Nếu một mục không hợp lệ được tìm thấy, hàm sẽ loại bỏ mục đó khỏi danh sách **entries** bằng cách sử dụng hàm **erase**. Đồng thời, vì một phần tử đã bị loại bỏ, biến lặp lại **i** sẽ giảm đi 1 để xử lý tiếp tục các phần tử tiếp theo trong danh sách.
* Hàm tiếp tục duyệt qua các mục khác trong danh sách và lặp lại quá trình kiểm tra và loại bỏ nếu cần.
* Khi đã duyệt qua tất cả các mục, danh sách **entries** sẽ không còn chứa các mục không hợp lệ.

## convertStringToLittleEdian

Chuyển đổi một chuỗi hexadeximal từ định dạng big-endian sang little-endian.



* Khởi tạo một chuỗi rỗng **result** để lưu trữ kết quả cuối cùng.
* Khởi tạo một bản sao của chuỗi đầu vào để không thay đổi chuỗi gốc (**copy**).
* Lặp qua chuỗi **copy** cho đến khi nó trở thành chuỗi rỗng:
  1. Trong mỗi vòng lặp, lấy hai ký tự đầu tiên từ **copy** và thêm chúng vào vector **temp**. Sau đó, loại bỏ hai ký tự này khỏi **copy**.
* Duyệt qua vector **temp** theo thứ tự ngược lại và thêm các phần tử vào chuỗi **result**.
* Trả về chuỗi **result** đã chuyển đổi.

## getStringFromVector

Hàm **getStringFromVector** được thiết kế để tạo ra một chuỗi từ một vector các chuỗi, bắt đầu từ vị trí **start** và lấy **length** phần tử tiếp theo trong vector.

A computer screen shot of a code

Description automatically generated

Cách thức hoạt động của hàm này:

* Khởi tạo một chuỗi rỗng **result** để lưu trữ kết quả cuối cùng.
* Duyệt qua các phần tử của vector **storedValues** từ vị trí **start** cho đến vị trí **start + length - 1**.
  + Trong mỗi vòng lặp, thêm giá trị của phần tử hiện tại vào chuỗi **result**.
* Trả về chuỗi **result** đã tạo.