

CSC10007 - Hệ điều hành

Project 01 - Quản lý hệ thống tập tin

Ngày 18 tháng 11 năm 2021

Mục lục

1	Thông tin	2
1.1	Danh sách thành viên	2
1.2	Bảng phân công công việc	2
1.3	Đánh giá tiến độ hoàn thành công việc	2
2	Các bước thực hiện	2
2.1	Master Boot Record	2
2.2	FAT32	5
2.2.1	Archive	19
2.2.2	Subdirectory	19
2.2.3	In ra kết quả	20
2.3	NTFS	21
3	Demo	28
3.1	Kết quả chạy chương trình để test Master Boot Record	28
3.1.1	Thông tin trên Master Boot Record	28
3.1.2	Thông tin trên bootable Boot Sector khi sử dụng thông tin từ Master Boot Record để đọc	29
3.2	Kết quả chạy chương trình trên USB FAT32	30
3.2.1	Thông tin trên Boot Sector	30
3.2.2	Cây thư mục trên ổ đĩa	31
3.3	Kết quả chạy chương trình trên ổ đĩa định dạng NTFS	32

1 Thông tin

1.1 Danh sách thành viên

MSSV	Họ và tên
19120338	Trần Hoàng Quân
19120383	Huỳnh Tấn Thọ
19120407	Lâm Hải Triều
19120426	Phan Đăng Diễm Uyên
19120469	Sử Nhật Đăng

1.2 Bảng phân công công việc

MSSV	Công việc
19120338	Viết + kiểm thử phần NTFS, viết báo cáo
19120383	Viết + kiểm thử phần FAT32 + NTFS
19120407	Viết, kiểm thử phần NTFS, viết báo cáo
19120426	Viết, kiểm thử FAT32, viết báo cáo
19120469	Viết, kiểm thử phần FAT32 + Master Boot Record

1.3 Đánh giá tiến độ hoàn thành công việc

Công việc	Đánh giá
FAT32	Đọc và hiển thị được cây thư mục cùng các thông tin liên quan đến Boot Sector.
NTFS	Đọc và hiển thị được tên tập tin + trạng thái + kích thước + nội dung tập tin.
Master Boot Record	Đọc và hiển thị được nội dung thông Master Boot Record và các Partion Table

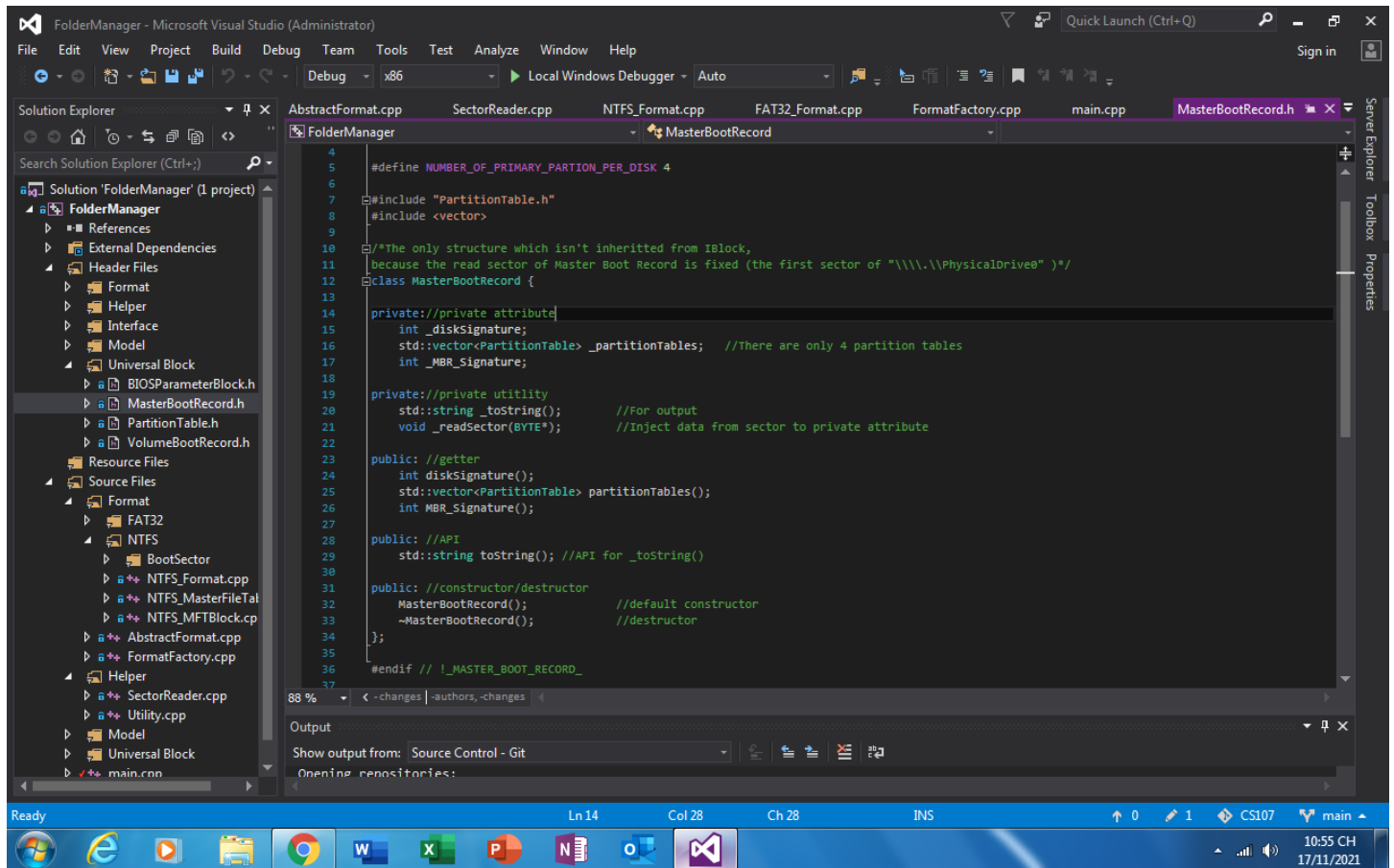
2 Các bước thực hiện

2.1 Master Boot Record

Master Boot Record được đọc từ 512 byte đầu tiên của sector 0 trên ổ đĩa, được cài đặt trong MasterBootRecord.h:

Nội dung	Kích thước	Offset
Code Area	440 byte	0x00
Optional disk signature	4 byte	0x1B8
4 x Partition tables	4 x 16 byte	0x01BE
MBR signature	2 byte	0x01FE

Bảng 1: Nội dung Master Boot Record

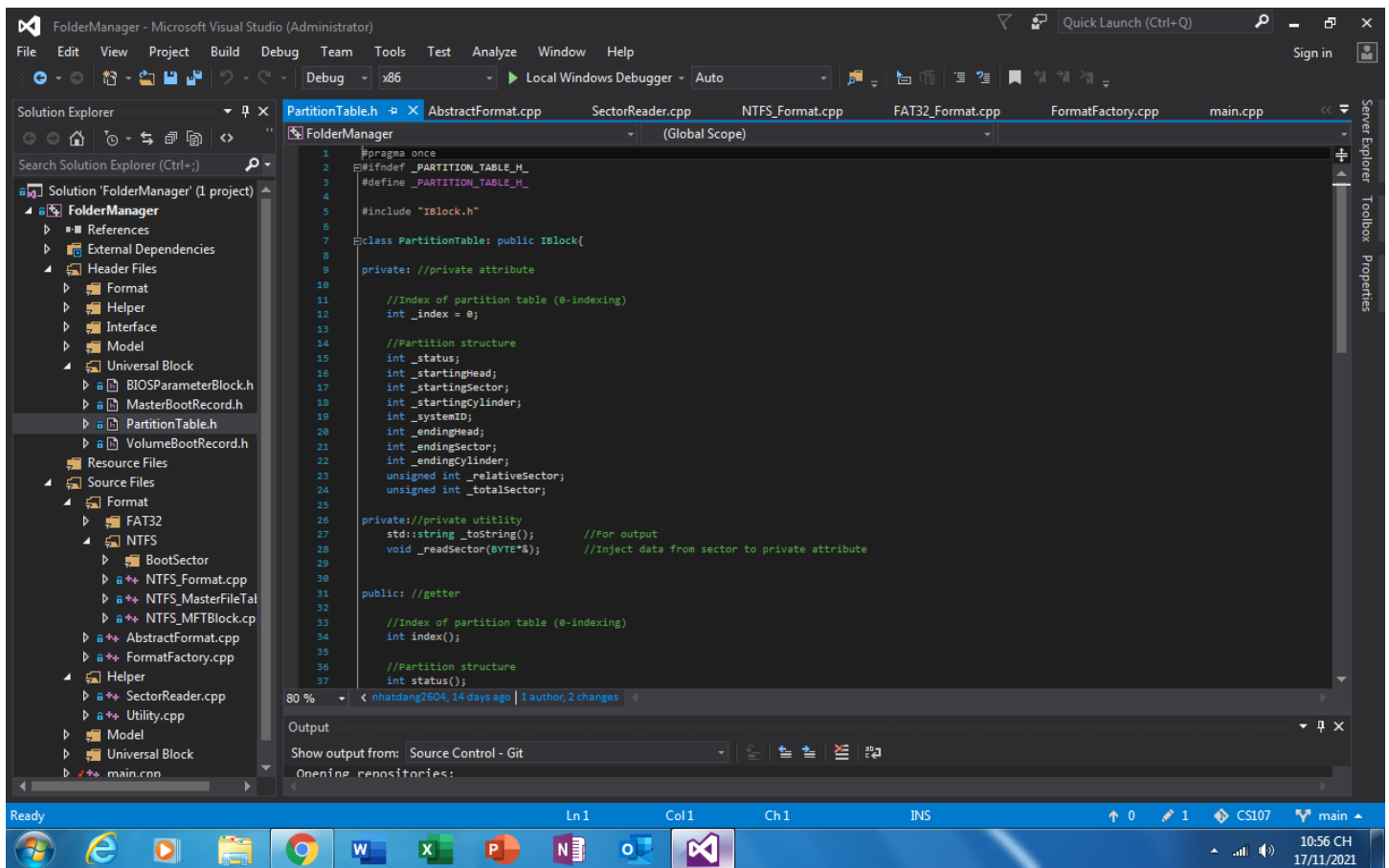


Hình 1: Thuộc tính class MBR

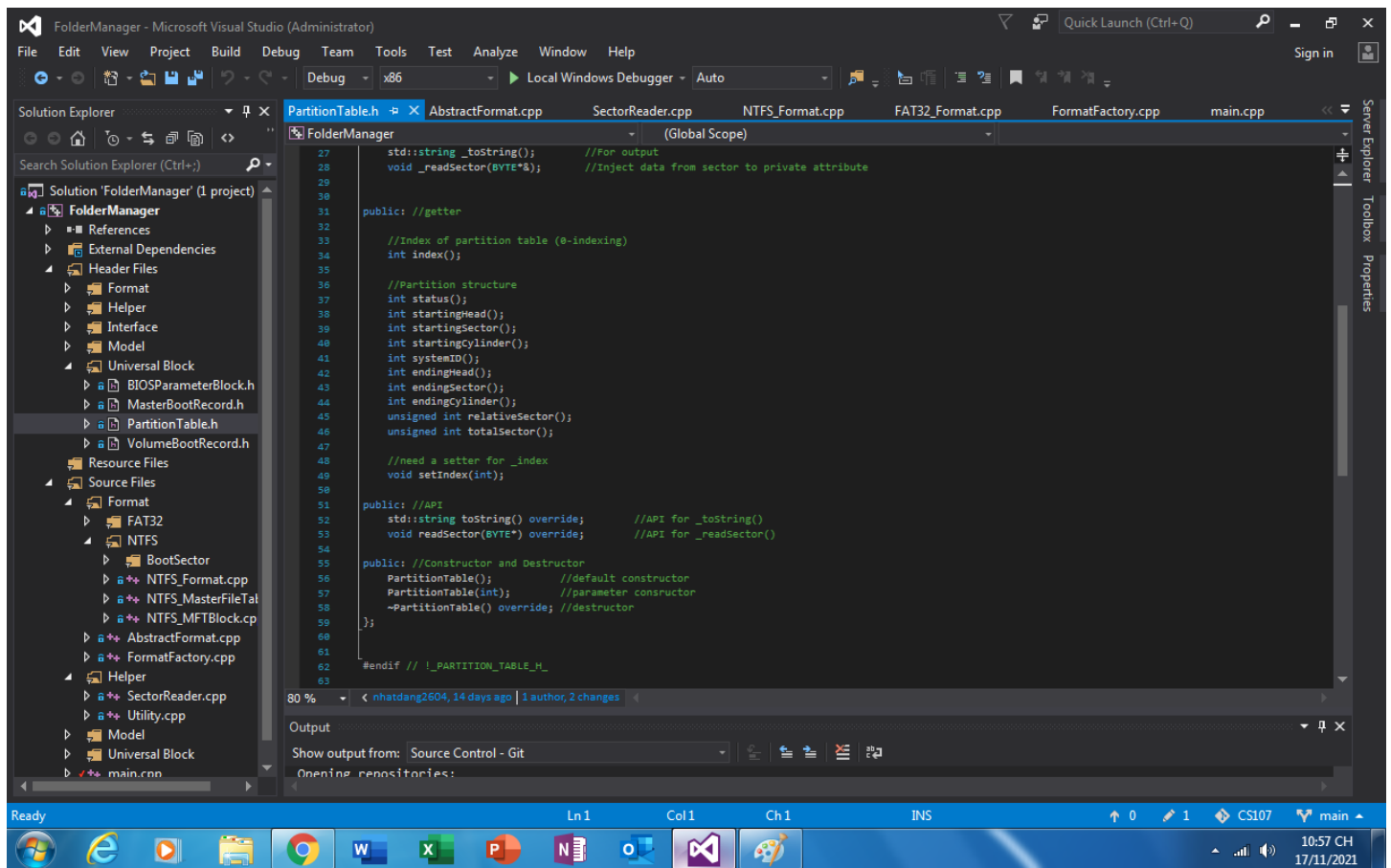
Ta có thể thấy được trong cấu trúc của Master Boot Record có 4 partition table. Vì trong ổ cứng chỉ có tối đa 4 partition table này nên 1 ổ cứng chỉ có thể chia tối đa thành 4 primary partition. Partition Table được em cài đặt trong PartitionTable.h. Cấu trúc của Partition Table như sau:

Nội dung	Kích thước	Offset
Status	1 byte	0x00
Starting head	1 byte	0x01
Starting sector	1 byte	0x02
Starting cylinder	1 byte	0x03
Partition type	1 byte	0x04
Ending head	1 byte	0x05
Ending sector	1 byte	0x06
Ending cylinder	1 byte	0x07
Relative sector	4 byte	0x08
Total sector	4 byte	0x0C

Bảng 2: Nội dung Partition Table



Hình 2: Thuộc tính class Partition Table (1)



Hình 3: Thuộc tính class Partition Table (2)

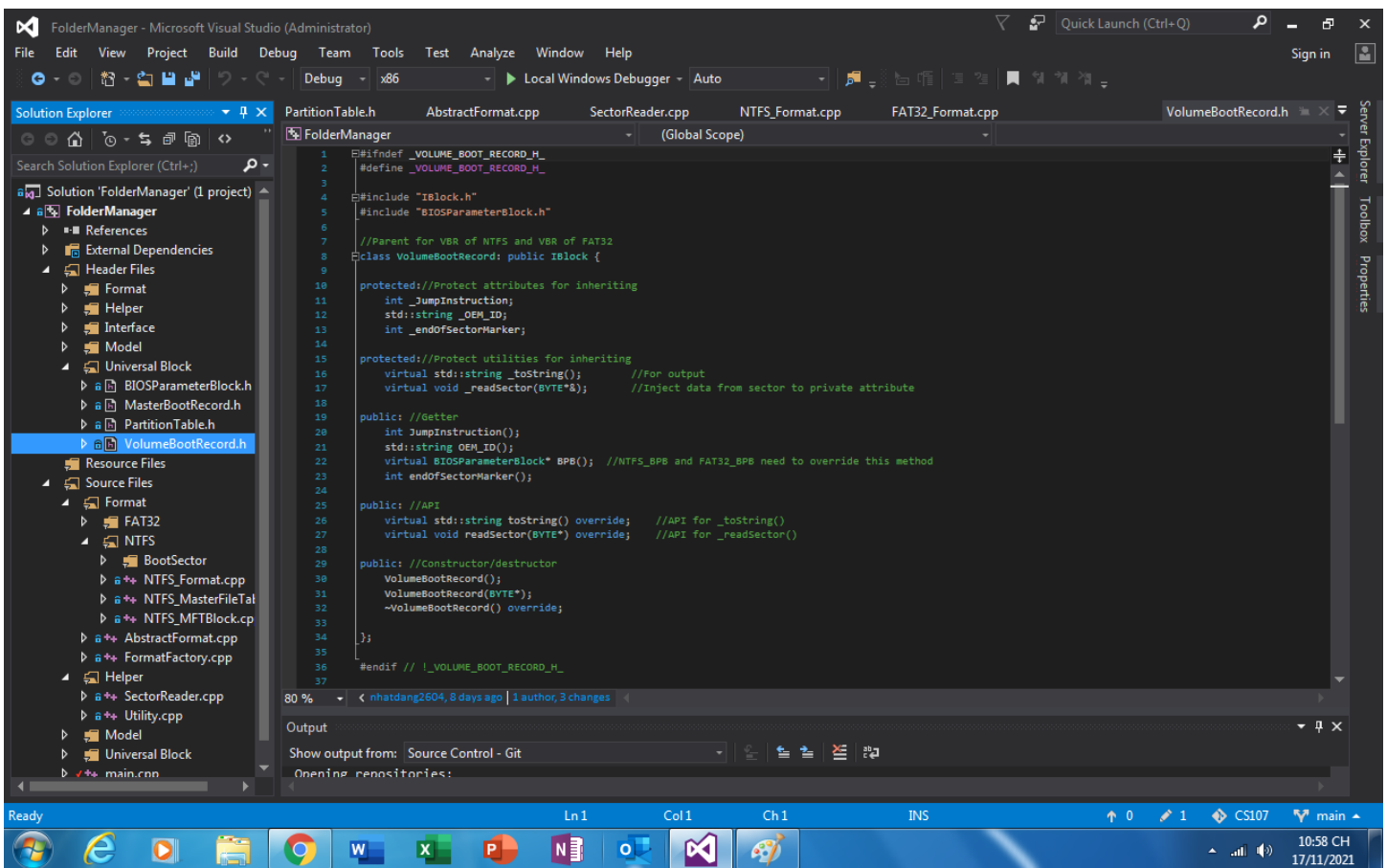
Ở một máy tính bình thường không sử dụng dual boot, sẽ có 1 partition table sao cho giá trị status mang giá trị là 0x80, mang ý nghĩa là bootable. Bằng việc đọc 512 byte kể từ trường relative sector của partition table này, ta sẽ dễ dàng có được boot sector của phân vùng ổ đĩa cài đặt hệ điều hành (ở Windows chính là ổ C:/)

2.2 FAT32

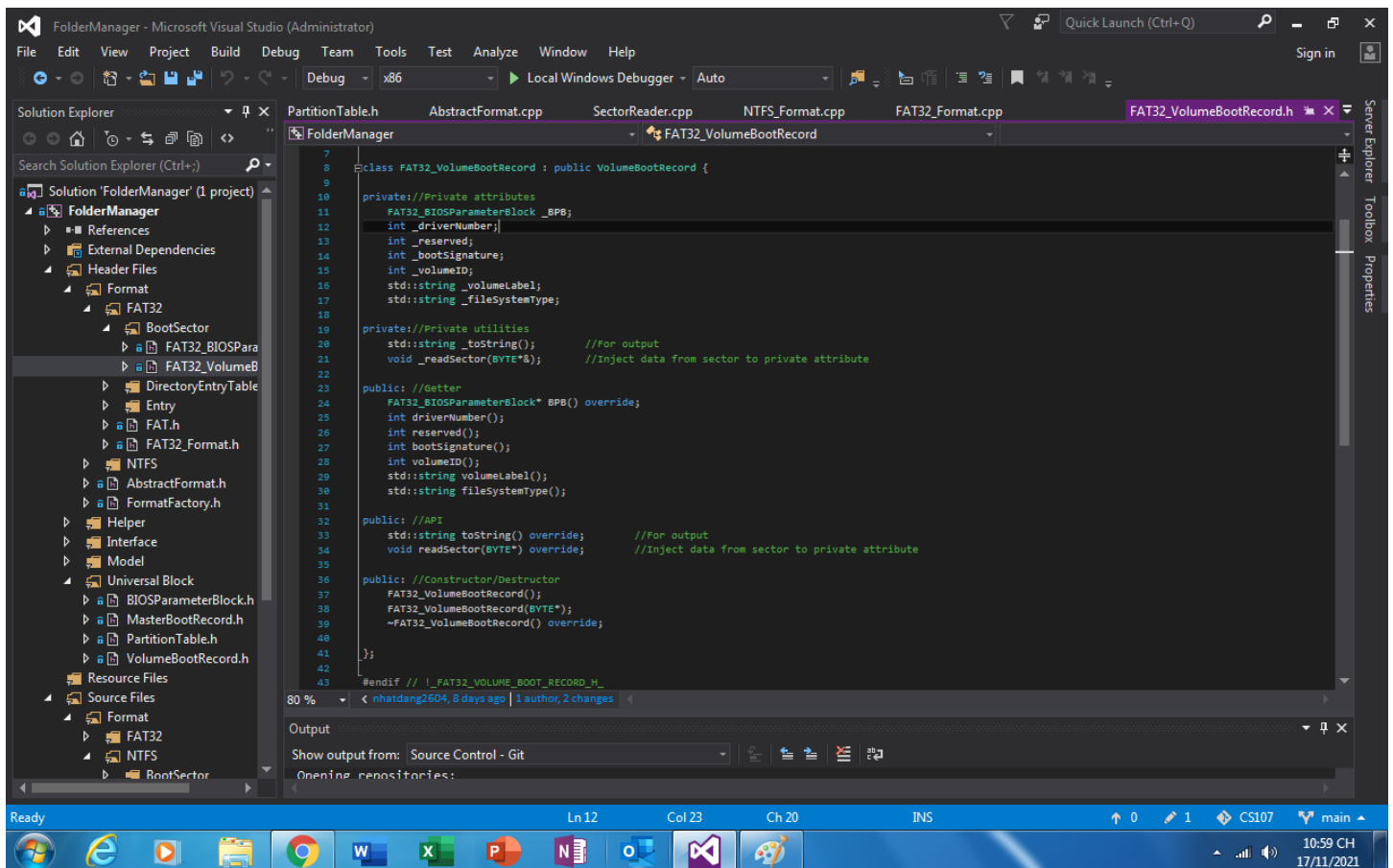
Volume Boot Record được đọc từ 512 byte đầu của sector 0 trên phân vùng ổ đĩa đang đọc, được cài đặt trong file `FAT32_VolumeBootRecord.h`. `FAT32_VolumeBootRecord` được kế thừa từ `VolumeBootRecord`

Nội dung	Kích thước	Offset
Jump instruction	3 byte	0x00
OEM ID	8 byte	0x03
Drive number	1 byte	0x40
Reserved	1 byte	0x41
Boot signature	1 byte	0x42
Volume ID	4 byte	0x43
Volume label	11 byte	0x47
File system type	8 byte	0x52
End of sector marker	2 byte	0x01FE

Bảng 3: Nội dung VBR



Hình 4: Thuộc tính class Volume Boot Record

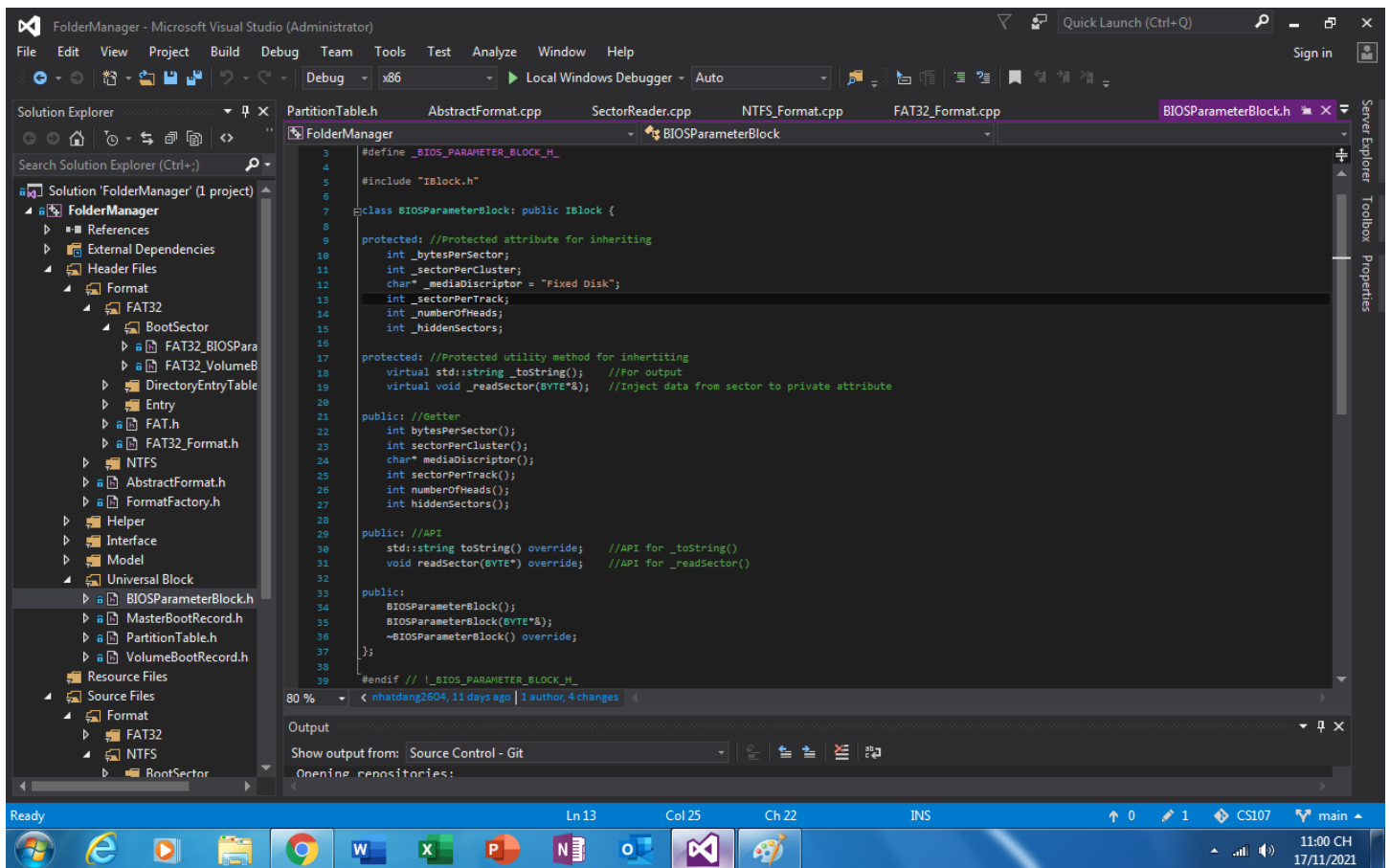


Hình 5: Thuộc tính class FAT32 Volume Boot Record

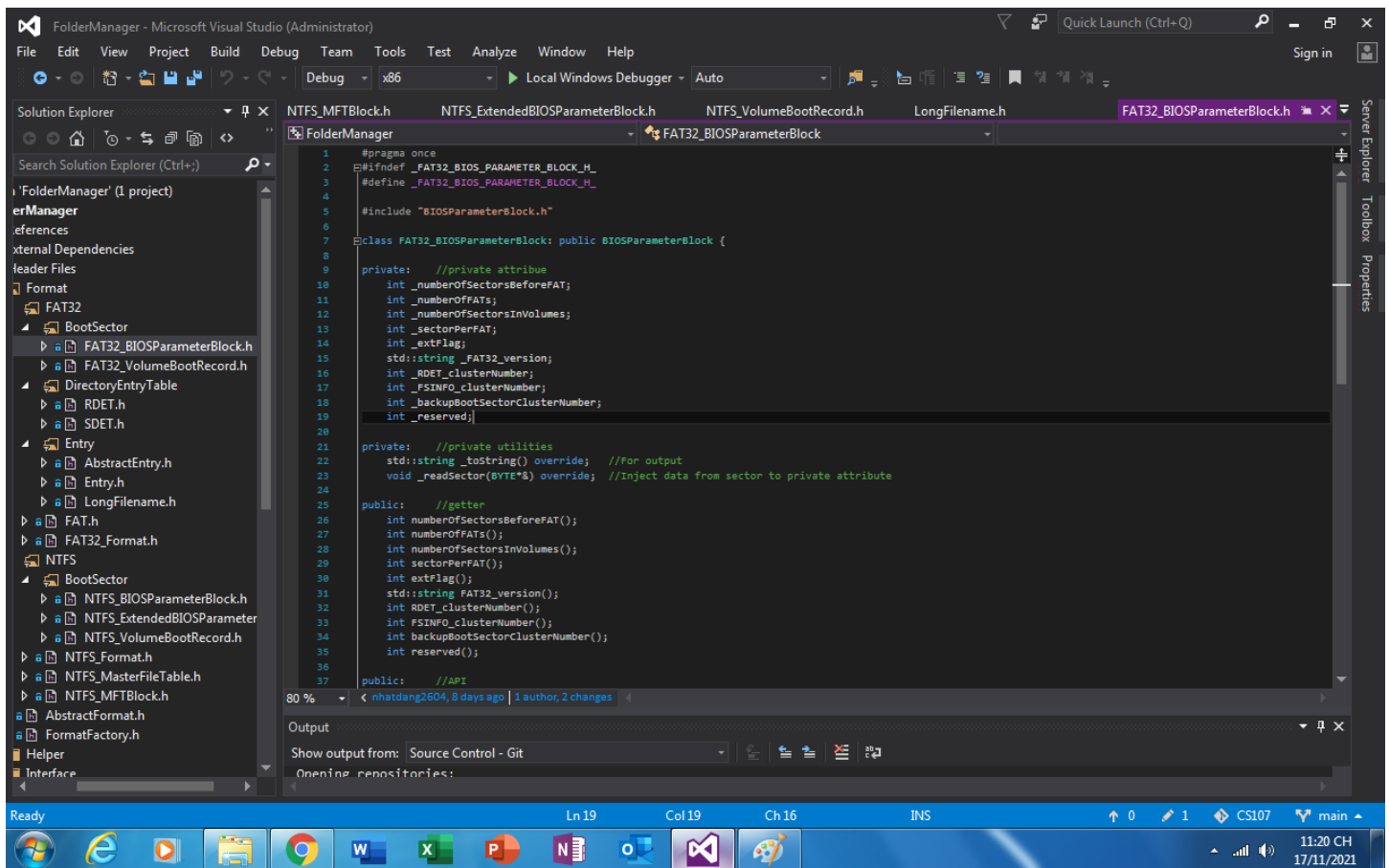
Các phần BPB được đọc như sau, BPB được cài đặt trong FAT32_BIOSParameterBlock.h. FAT32_BIOSParameterBlock được kế thừa từ BIOSParameterBlock

Nội dung	Kích thước	Offset
Bytes per sector	2 byte	0x0B
Sector per cluster	1 byte	0x0D
Sector per track	2 byte	0x18
Number of heads	2 byte	0x1A
Hidden sectors	4 byte	0x1C
Number of sector before FAT	2 byte	0x0E
Number of FATs	1 byte	0x10
Number of sectors in Volumes	4 byte	0x20
Sector per FAT	4 byte	0x24
Ext Flag	2 byte	0x28
FAT32 version: minor	1 byte	0x2B
FAT32 version: major	1 byte	0x2A
RDET cluster number	4 byte	0x2C
FSINFO cluster number	2 byte	0x30
Backup Boot Sector cluster number	2 byte	0x32
Reserved	12 byte	0x34

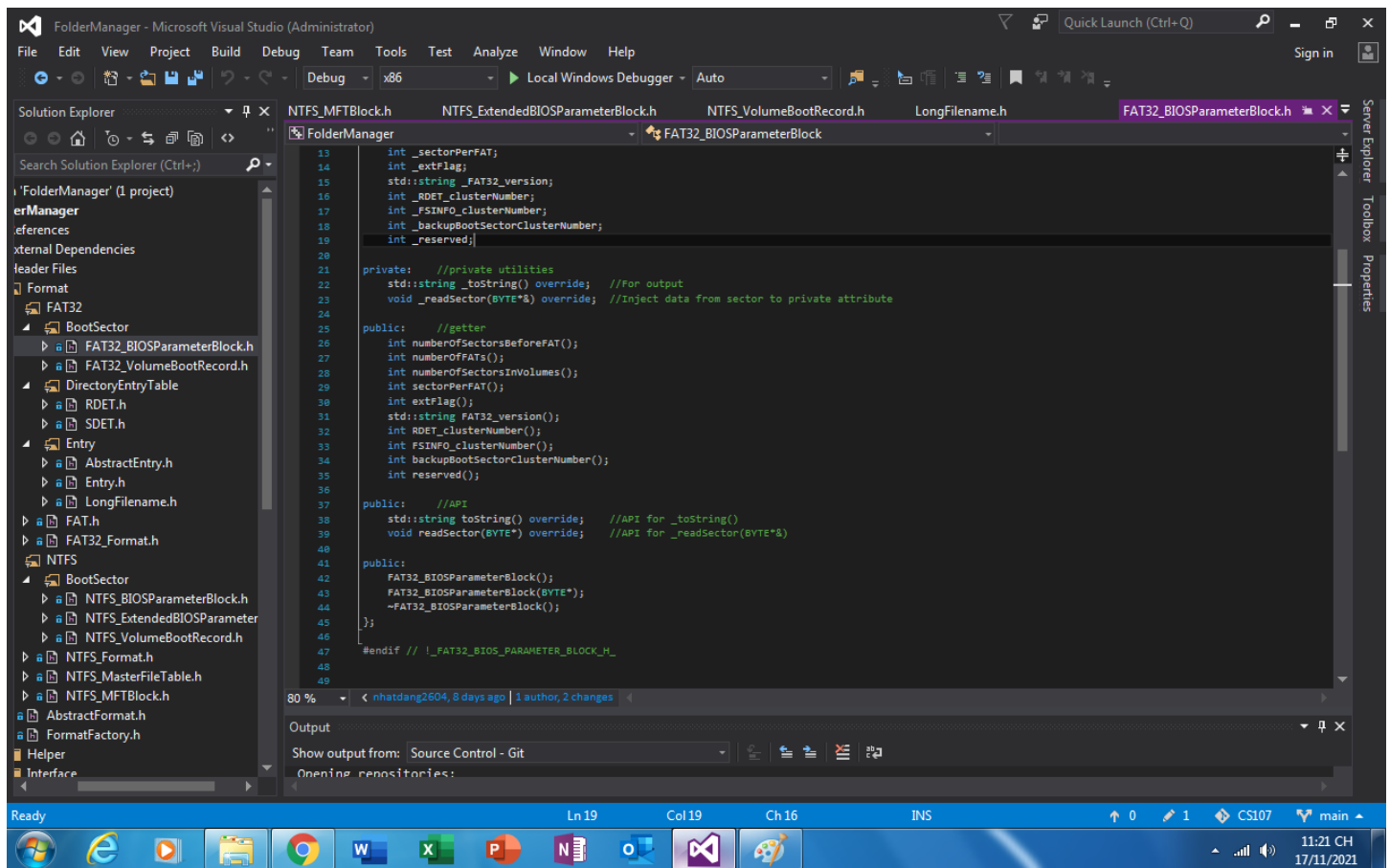
Bảng 4: Nội dung BPB



Hình 6: Thuộc tính class BIOS Parameter Block

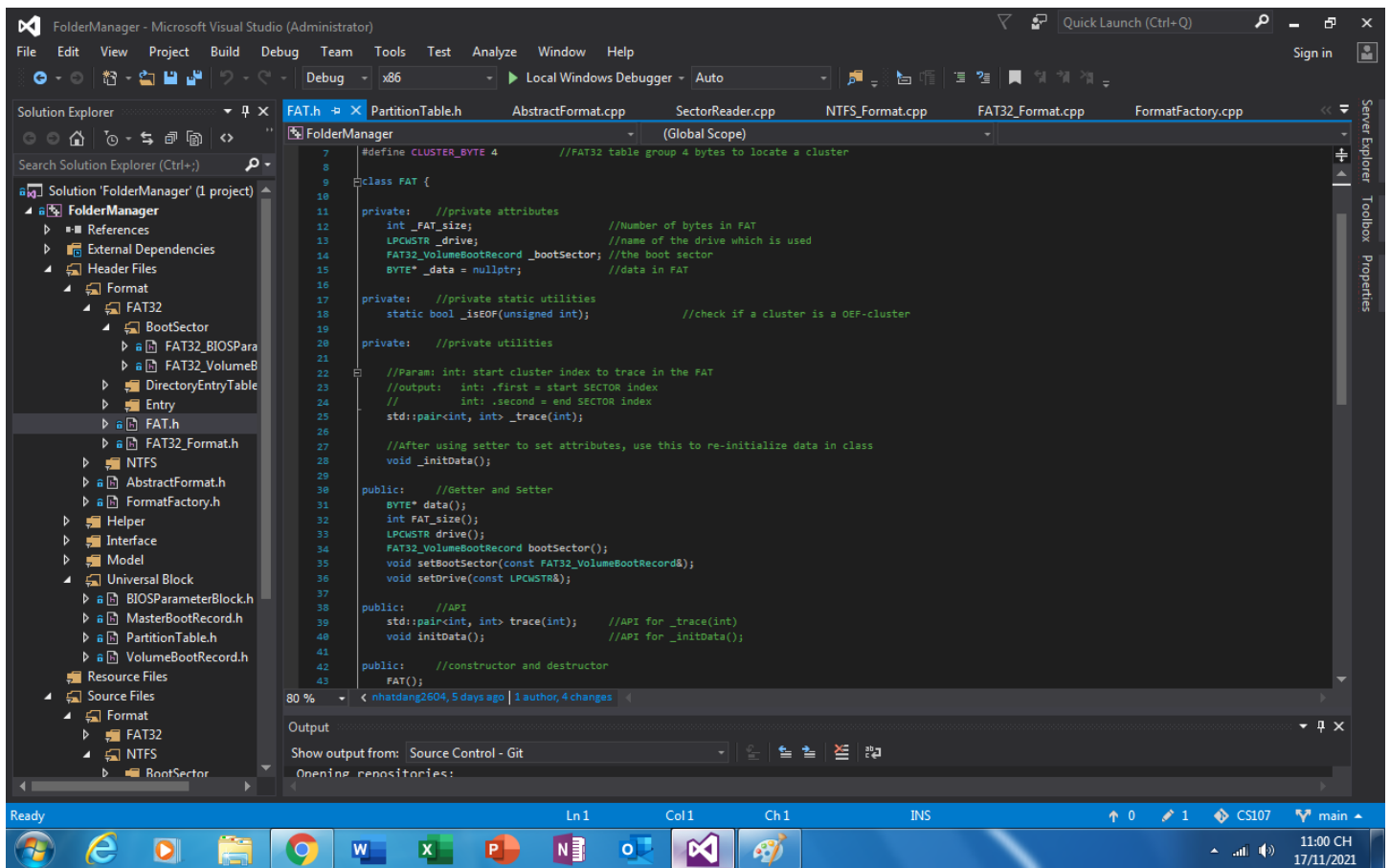


Hình 7: Thuộc tính class FAT32 BIOS Parameter Block (1)

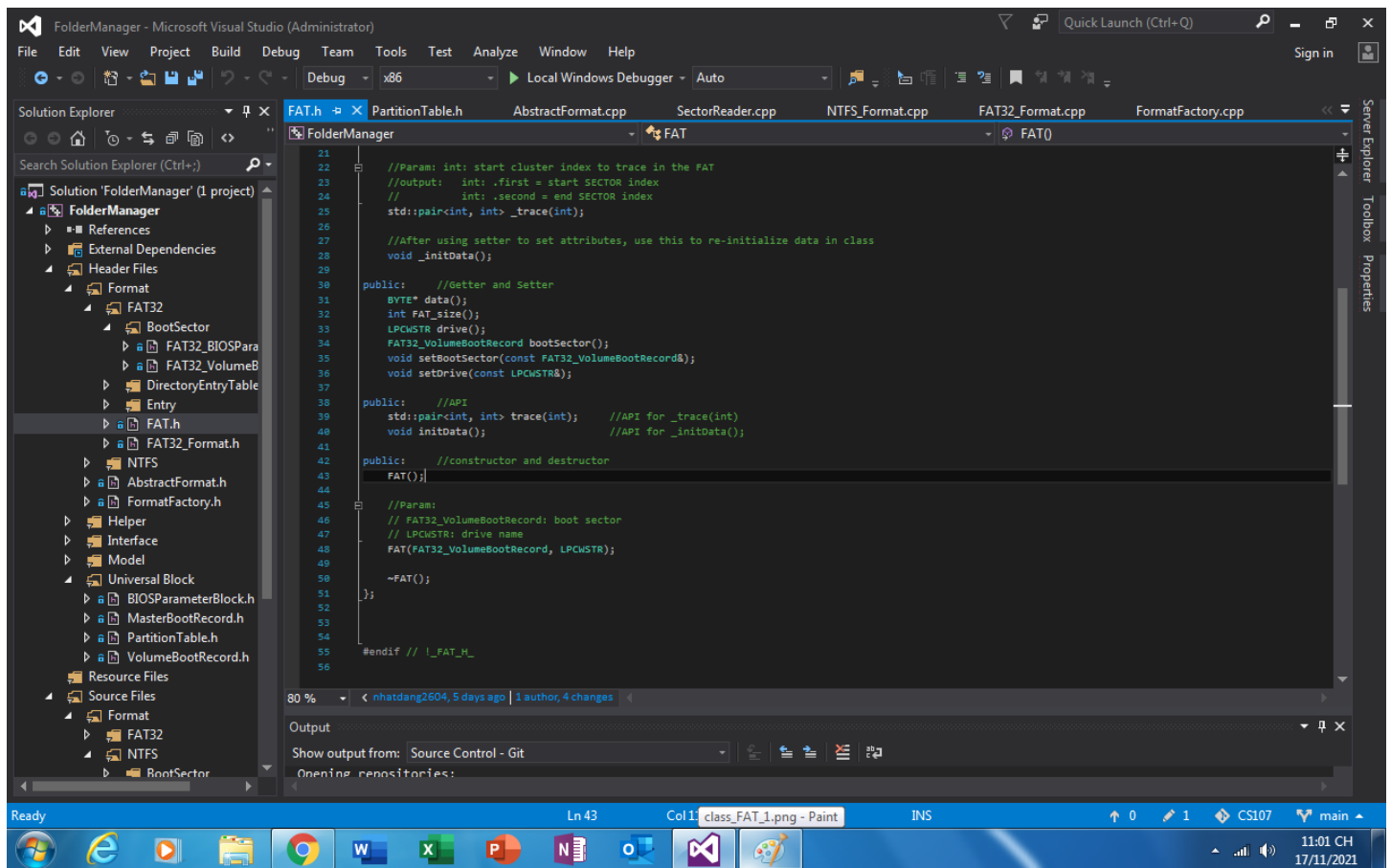


Hình 8: Thuộc tính class FAT32 BIOS Parameter Block (2)

- Sau khi có được trường "Number of sector before FAT" và "Sector per FAT" kết hợp với "Bytes per sector", ta có thể dễ dàng tìm được vị trí để đọc bảng FAT và kích thước bảng FAT. Bảng FAT đọc được sẽ có nội dung thô (bởi vì thực chất bảng FAT cũng không có trường gì nên chúng em sẽ không đề cập tới, chỉ toàn là những bộ cluster gồm 4 byte đề dờ, đối với FAT32) được lưu trong mảng `BYTE* _data` cài đặt trong `FAT.h`.

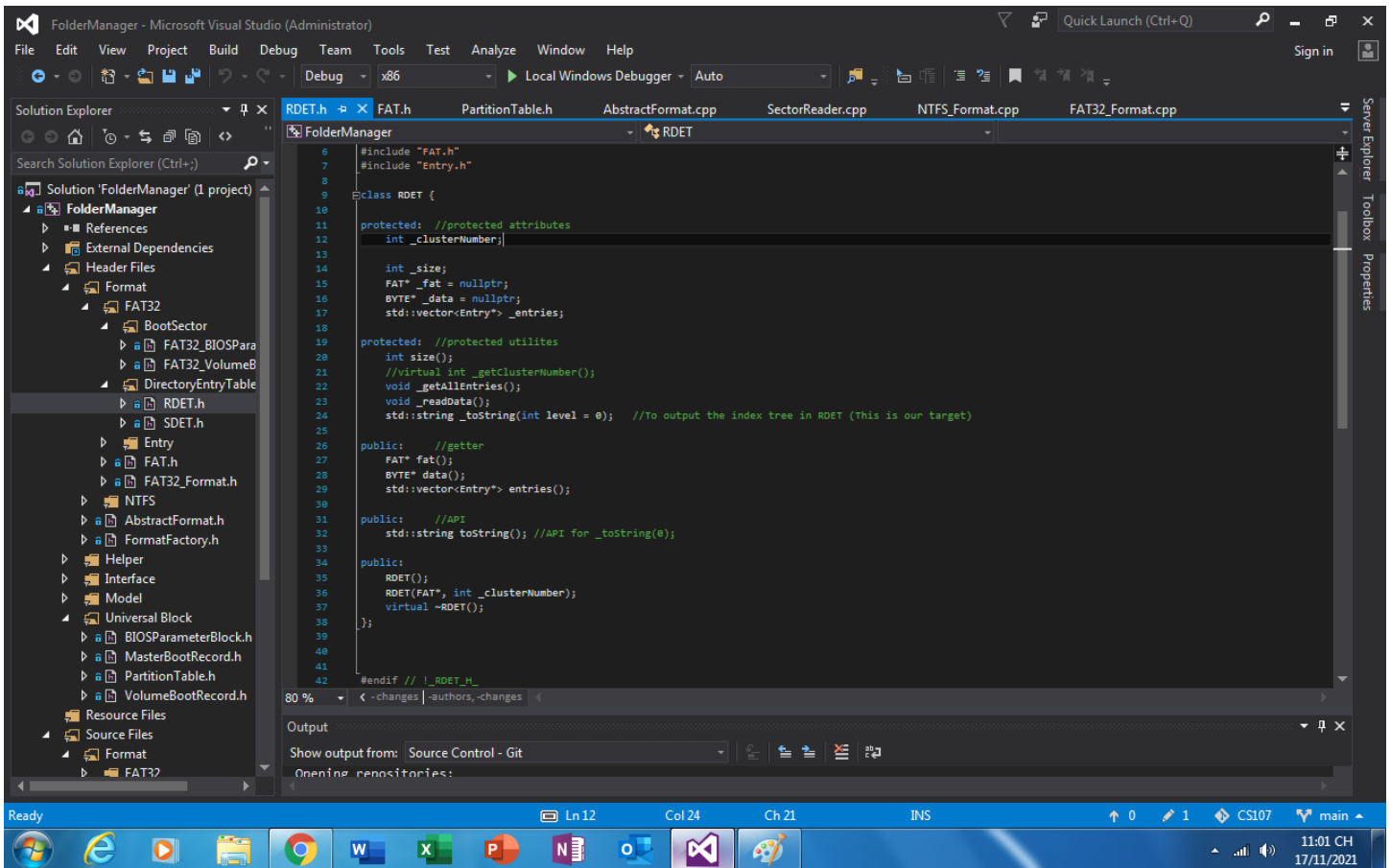


Hình 9: Thuộc tính class FAT (1)



Hình 10: Thuộc tính class FAT (2)

- Tiếp theo, sau khi có được nội dung bảng FAT, ta tiến hành dò trên bảng FAT để tìm kiếm RDET. Trên BPB có 1 trường là "RDET cluster number", trường này chính là vị trí của cluster của RDET khi dò trên bảng FAT. Ở máy em, trường này có giá trị là 2. Sau khi dò được trên nội dung trường FAT, ta cũng sẽ có kích thước của RDET. RDET được cài đặt trong RDET.h. RDET chẳng có cấu trúc gì ngoài trừ nhiều entry với kích thước 32 byte nối tiếp nhau, nên em sẽ không đề cập tới cấu trúc của RDET mà chỉ quan tâm tới entry.

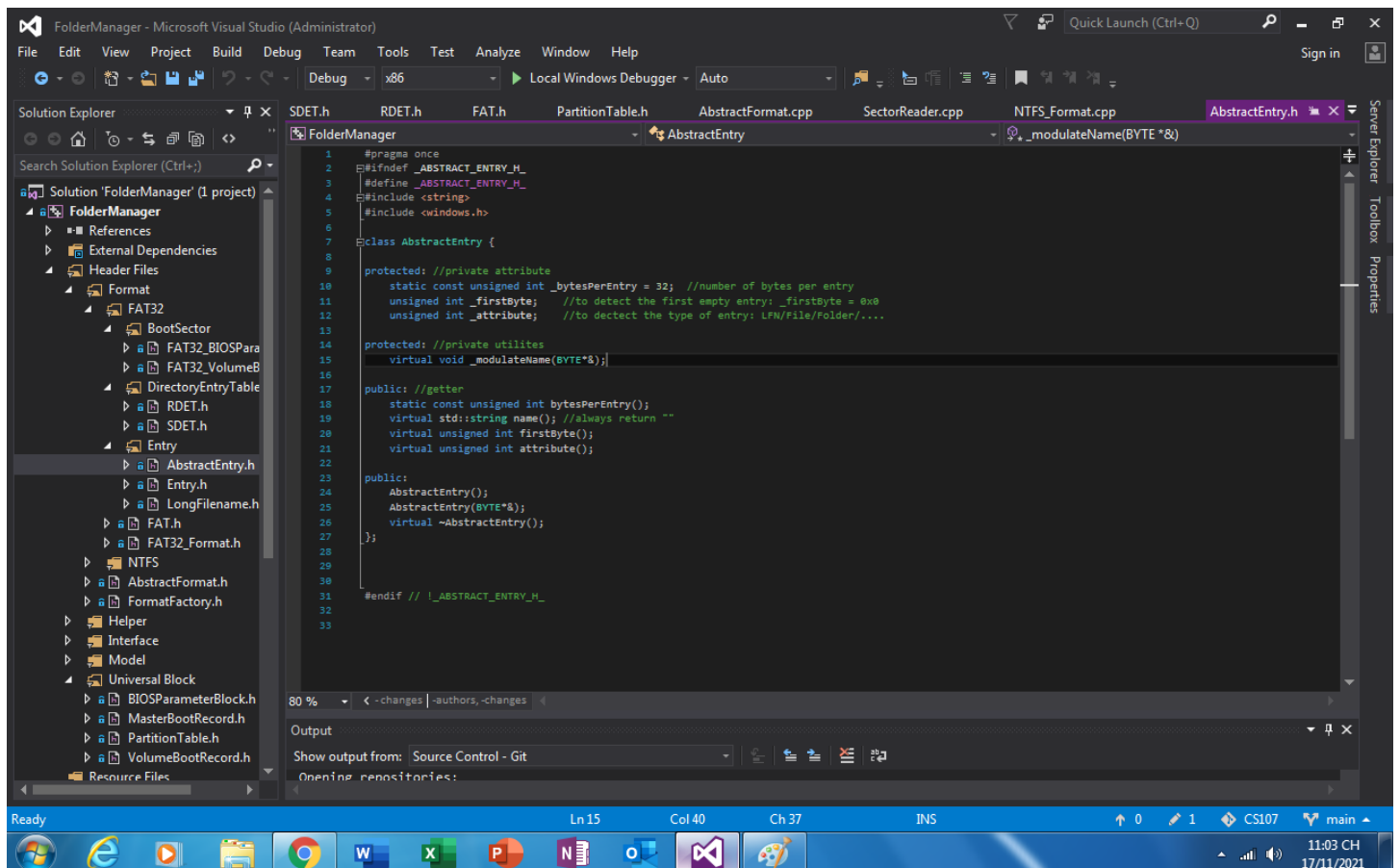


Hình 11: Thuộc tính class RDET

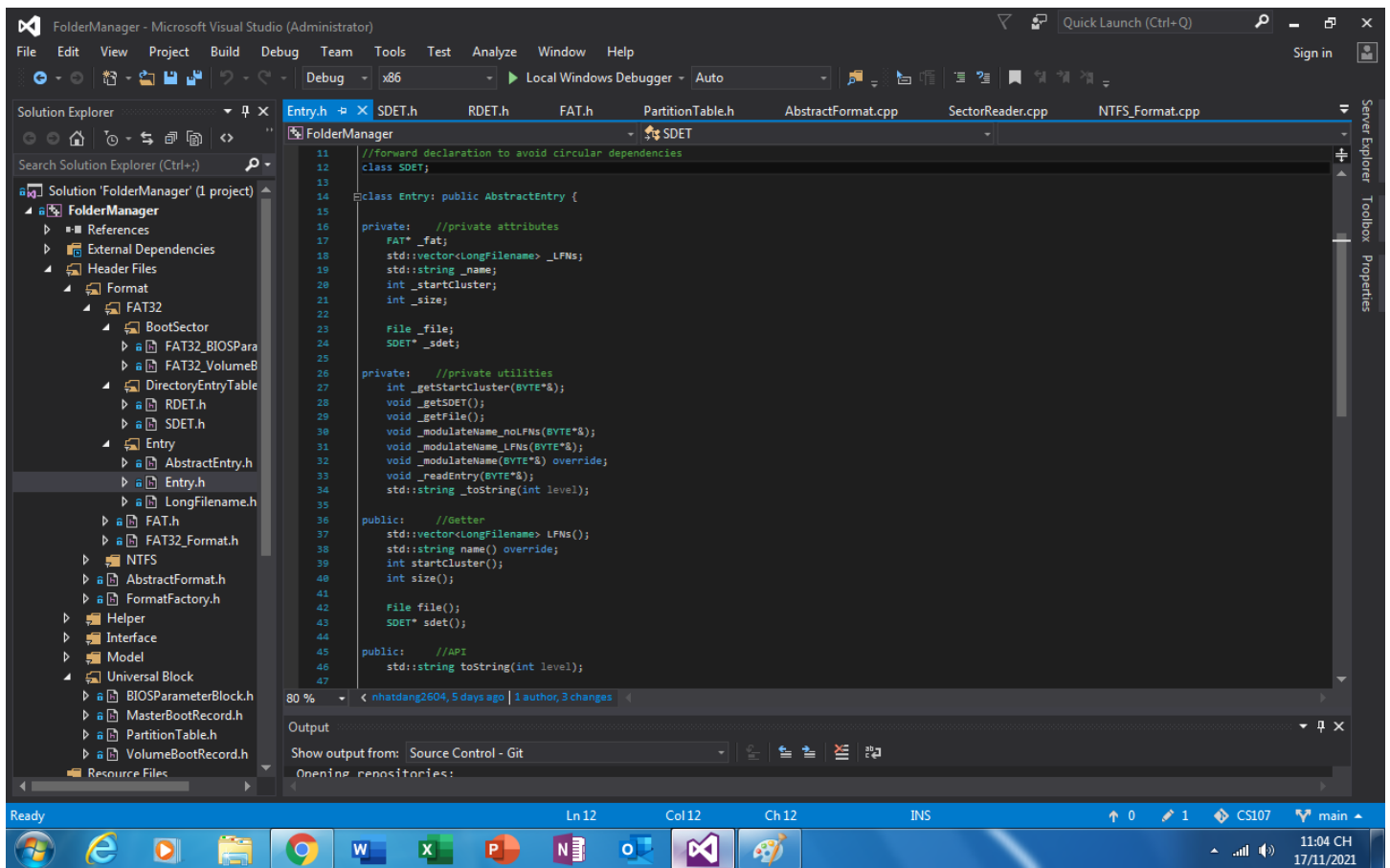
- **Lưu ý:** khi đọc entry, ta cần phải bỏ qua những entry bị xóa: những Entry có byte đầu tiên là 0xE5
- Với mỗi cluster của RDET, ta sẽ có số entry trong RDET là: Bytes per sector * Sector per cluster / 32. Entry có 2 loại: chính và phụ. Entry chính được cài đặt trong Entry.h. Entry được kế thừa từ AbstractEntry

Nội dung	Kích thước	Offset
DIR_Name	11 byte	0x00
DIR_Attr	1 byte	0x0B
DIR_NTRes	1 byte	0x0C
DIR_CrtTimeTenth	1 byte	0x0D
DIR_CrtTime	2 byte	0x0E
DIR_CrtDate	2 byte	0x10
DIR_LstAccDate	2 byte	0x12
DIR_FstClusHI	2 byte	0x14
DIR_WrtTime	2 byte	0x16
DIR_WrtDate	2 byte	0x18
DIR_FstClusLO	2 byte	0x1A
DIR_FileSize	4 byte	0x1C

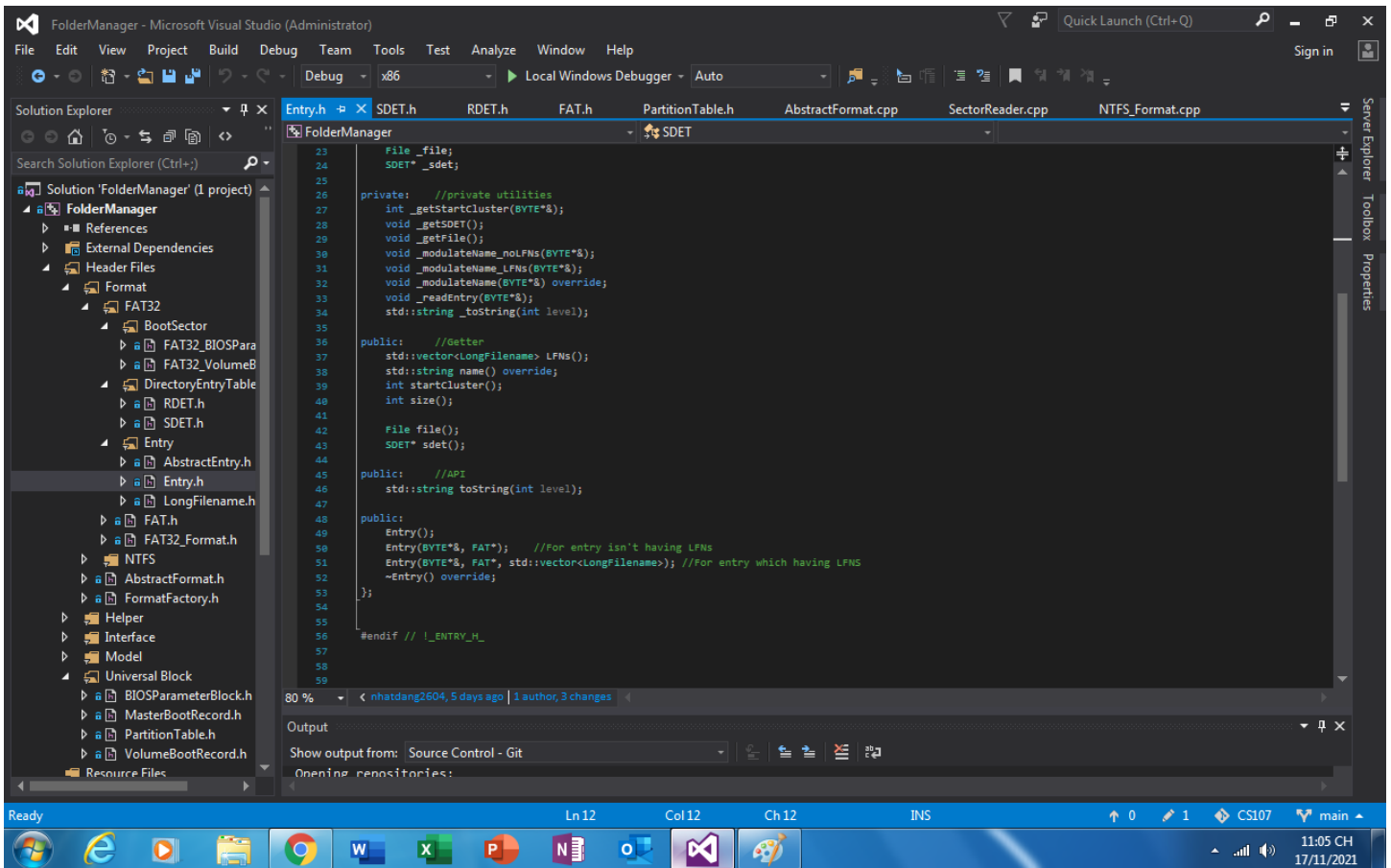
Bảng 5: Nội dung Entry chính



Hình 12: Thuộc tính class Abstract Entry



Hình 13: Thuộc tính class Entry (1)

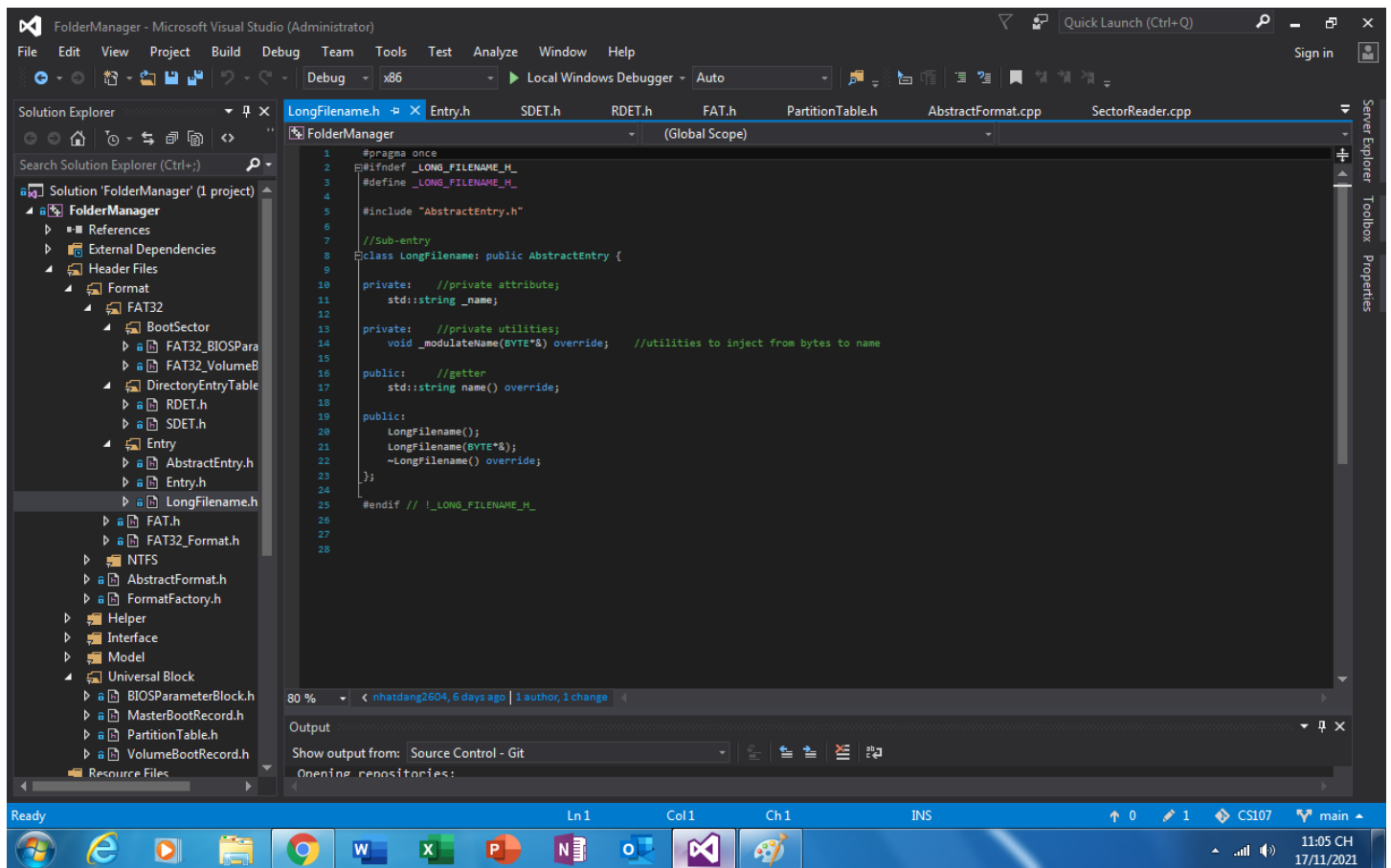


Hình 14: Thuộc tính class Entry (2)

Entry phụ được cài đặt trong LongFilename.h. LongFilename cũng được kế thừa từ AbstractEntry

Nội dung	Kích thước	Offset
LDIR_Ord	1 byte	0x00
LDIR_Name1	10 byte	0x01
LDIR_Attr	1 byte	0x0B
LDIR_Type	1 byte	0x0C
LDIR_Chksum	1 byte	0x0D
LDIR_Name2	12 byte	0x0E
LDIR_FstClusLO	2 byte	0x1A
LDIR_Name3	4 byte	0x1C

Bảng 6: Nội dung Entry phụ



Hình 15: Thuộc tính class Long Filename

Để phân biệt được liệu 32 byte entry ta đang đọc được là entry chính hay entry phụ, ta nhìn vào 1 byte từ offset 0x0B ở entry ta đang đọc và kiểm tra liệu giá trị có phải là 0x0F hay không, nếu đúng thì đó là entry phụ. Ngược lại là entry chính. Mặt khác, nếu đây là entry chính, thì ta tạm thời chỉ quan tâm xem liệu đây là archived hay subdirectory, và nếu là archived có đuôi là .txt thì sẽ in ra nội dung của file này, vậy làm thế nào? Ta dò bảng sau đây ở giá trị bit bật ở 1 byte từ offset 0x0B, để xem tập tin hiện tại có thuộc tính gì

8n	Mask	Mô tả
0	0x01	Read only
1	0x02	Hidden
2	0x04	System
3	0x08	Volume label
4	0x10	Subdirectory
5	0x20	Archive
6	0x40	Device (internal use only)
7	0x80	Unused

Bảng 7: Thuộc tính của tập tin

- Tiếp theo, ta tiến hành kết hợp cái entry phụ vào entry chính để lấy tên của tập tin tương ứng. Ta thực hiện việc này bằng cách `.push()` các entry phụ vào 1 cái **stack** cho tới khi nào gặp entry chính. Và entry chính sẽ được nạp cái **stack** này vào (em sử dụng **vector** như 1 cái **stack** để dễ tùy biến hơn). Sau đấy tại cái **stack** này trên file, ta `.pop()` từng entry phụ ra và ghép tên từ từ cho đến khi **stack** rỗng, thì ta sẽ được tên tập tin theo đúng thứ tự.

Vậy là ta đã xác định xem được hiện tại tập tin là archive hay directory, lúc này ta sẽ chia làm 2 trường hợp

2.2.1 Archive

- **Nếu đây không phải là file .txt:** Ta chỉ cần đọc các thông tin về tên, trạng thái và kích thước trên RDET và dò các sector tương ứng của file đấy là xong, ta lưu trữ các dữ liệu này để sau khi hoàn thành việc đọc sẽ in ra hết

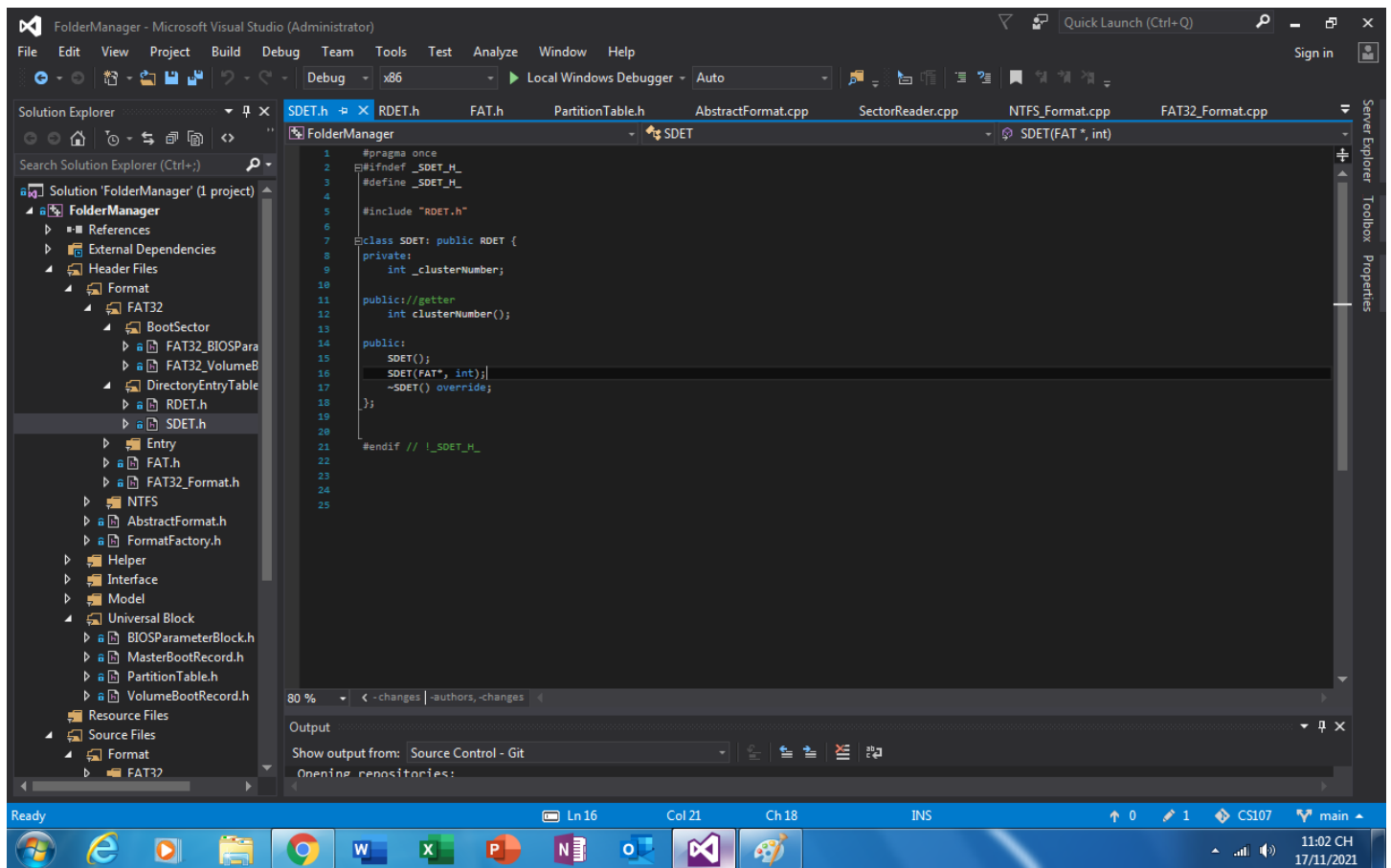
- **Nếu đây là file .txt:** Ta làm tương tự như đọc file không phải .txt, nhưng thêm vào 1 bước: đọc nội dung file .txt. Rất đơn giản, ta chỉ cần dò bảng FAT của file .txt này và di chuyển tới sector tương ứng. Sau khi đọc hết toàn bộ các sector này, ta chỉ cần rút gọn lại nội dung đã đọc được (vì ta đọc cố định 512x byte, trong khi file .txt chỉ có thể chứa <512x kí tự) và lưu lại thêm 1 trường nội dung cho file .txt là xong.

- Cả 2 kiểu trên sau khi đọc dữ liệu xong sẽ được nạp vào class **Entry**

2.2.2 Subdirectory

- Ở đây, ta sẽ xuất hiện 1 thành phần mới: SDET

- SDET có cấu trúc y hệt RDET, chỉ có điều những entry đầu tiên là dot entry, và ta cũng chẳng quan tâm những entry này lắm nên không thành vấn đề. Đây là lí do SDET em chỉ cho kế thừa RDET và thêm 1 thành phần để việc quản lí cluster number đơn giản hơn. SDET được em cài đặt trong **SDET.h**

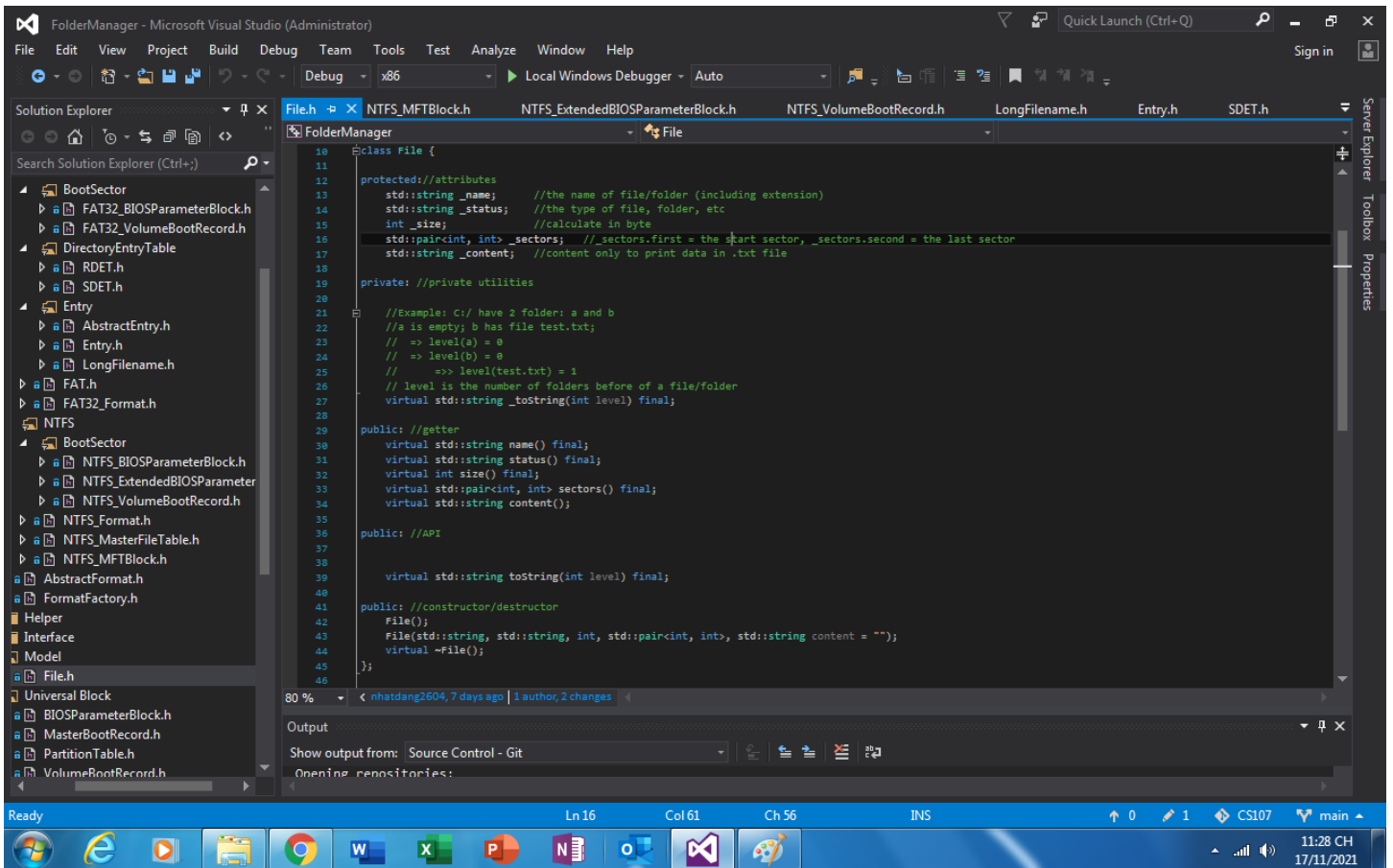


Hình 16: Thuộc tính class SDET

- Ta chỉ việc dò trên bảng FAT những cluster của subdirectory ta đang đọc, và đi tới những cluster tương ứng đây
- Tại đây, ta đọc toàn bộ những sector này vào SDET, **y hệt** như cách mà ta đã đọc RDET (đọc entry, dò entry trên bảng FAT để đọc archive/subdirectory trên chính SDET)
- Chính vì lí do trên, trong class **Entry** luôn có sẵn 1 pointer class **SDET** để lưu trữ SDET nếu là directory, nếu tập tin là archive thì pointer này sẽ mang giá trị **nullptr** - Sau khi đã đọc xong dữ liệu **bên trong** subdirectory, tất cả chúng sẽ được nạp vào SDET mà **Entry** của subdirectory đây nắm giữ, mặt khác các thông tin của bản thân subdirectory sẽ được nạp vào class **Entry**

2.2.3 In ra kết quả

- Các dữ liệu đã đọc được khi in ra sẽ được parse sang 1 model **File** đã được định nghĩa sẵn, tương ứng với mỗi tập tin đã đọc được. Các **File** sẽ được in ra màn hình bằng phương thức **.toString(int level)** với level là chỉ số cấp của cây thư mục. Tới đây là kết thúc việc đọc dữ liệu trên format FAT32



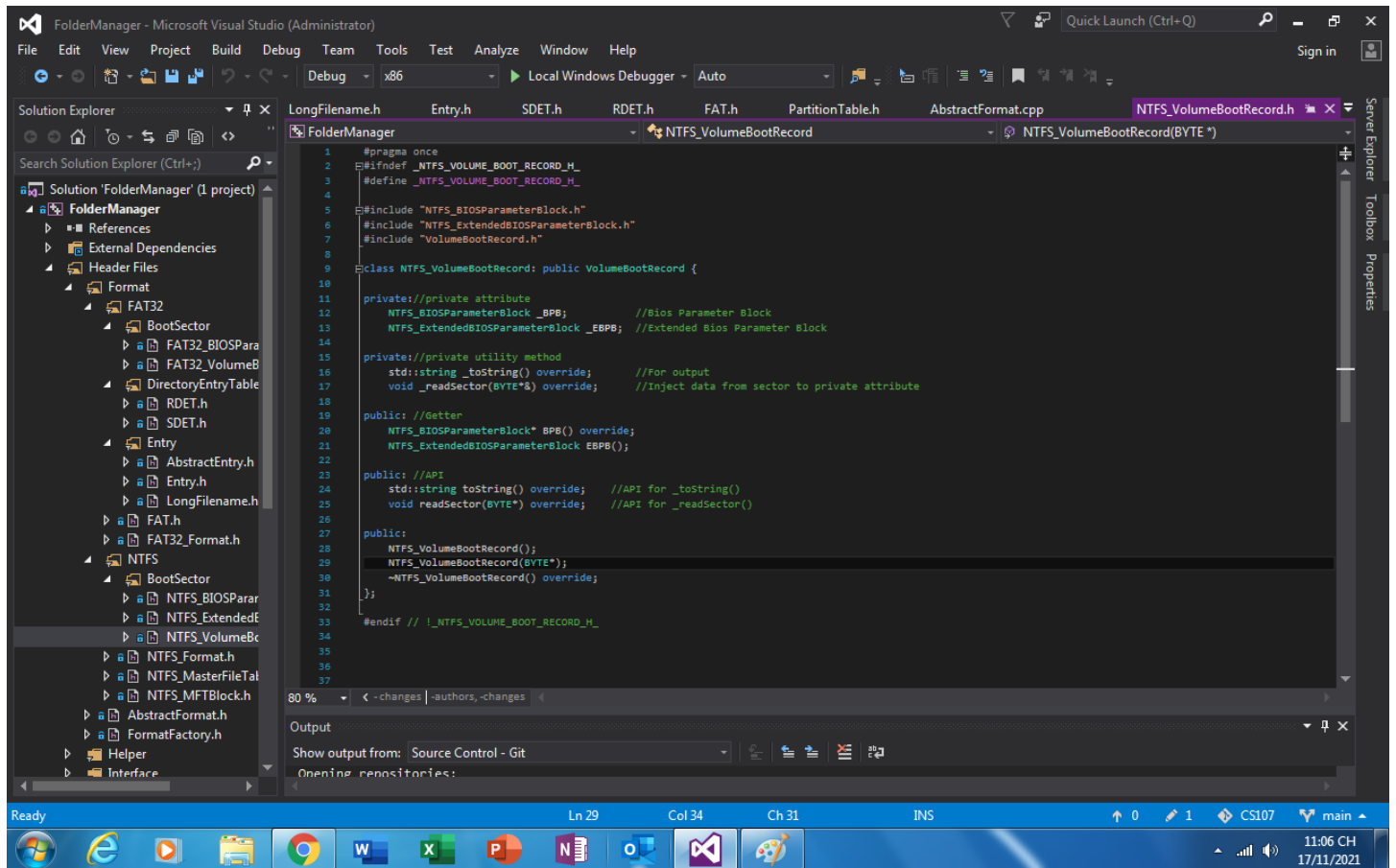
Hình 17: Thuộc tính class File

2.3 NTFS

Volume Boot Record được đọc từ 512 byte đầu của sector 0, được cài đặt trong file `NTFS_VolumeBootRecord.cpp`. `NTFS_VolumeBootRecord` được kế thừa từ `VolumeBootRecord`, vốn đã được đề cập ở mục **FAT32**.

Nội dung	Kích thước	Offset
Jump instruction	3 byte	0x00
OEM ID	8 byte	0x03
End of sector marker	2 byte	0x01FE

Bảng 8: Nội dung VBR

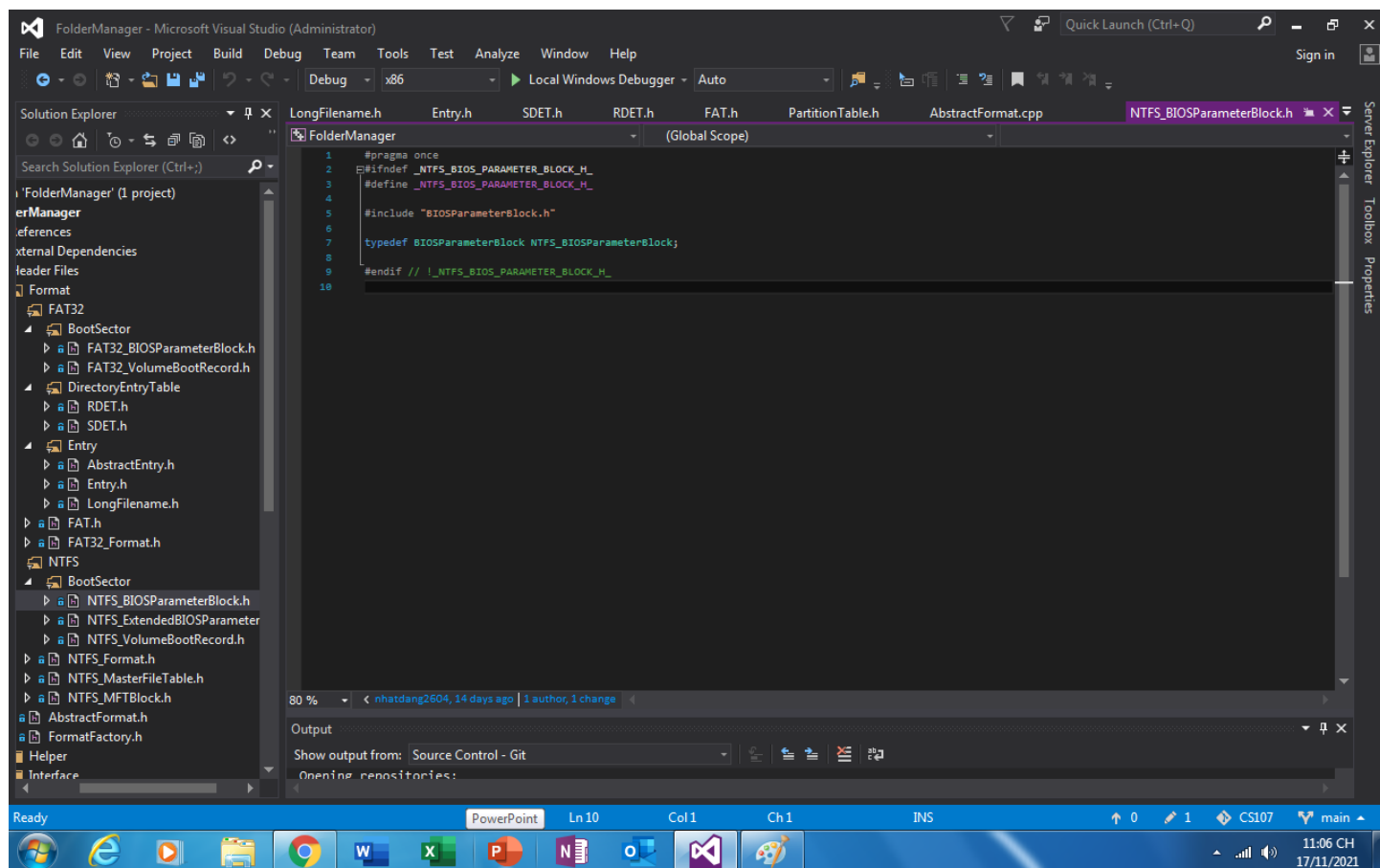


Hình 18: Thuộc tính class NTFS Volume Boot Record

Các phần BPB và EBPB được cài đặt lần lượt trong `NTFS_BIOSParameterBlock.h` và `NTFS_ExtendedBIOSParameterBlock.h`. `NTFS_BIOSParameterBlock` thực chất là `BIOSParameterBlock` (đã đề cập ở mục **FAT32**) đổi tên.

Nội dung	Kích thước	Offset
Bytes per sector	2 byte	0x0B
Sector per cluster	1 byte	0x0D
Sector per track	2 byte	0x18
Number of heads	2 byte	0x1A
Hidden sectors	4 byte	0x1C

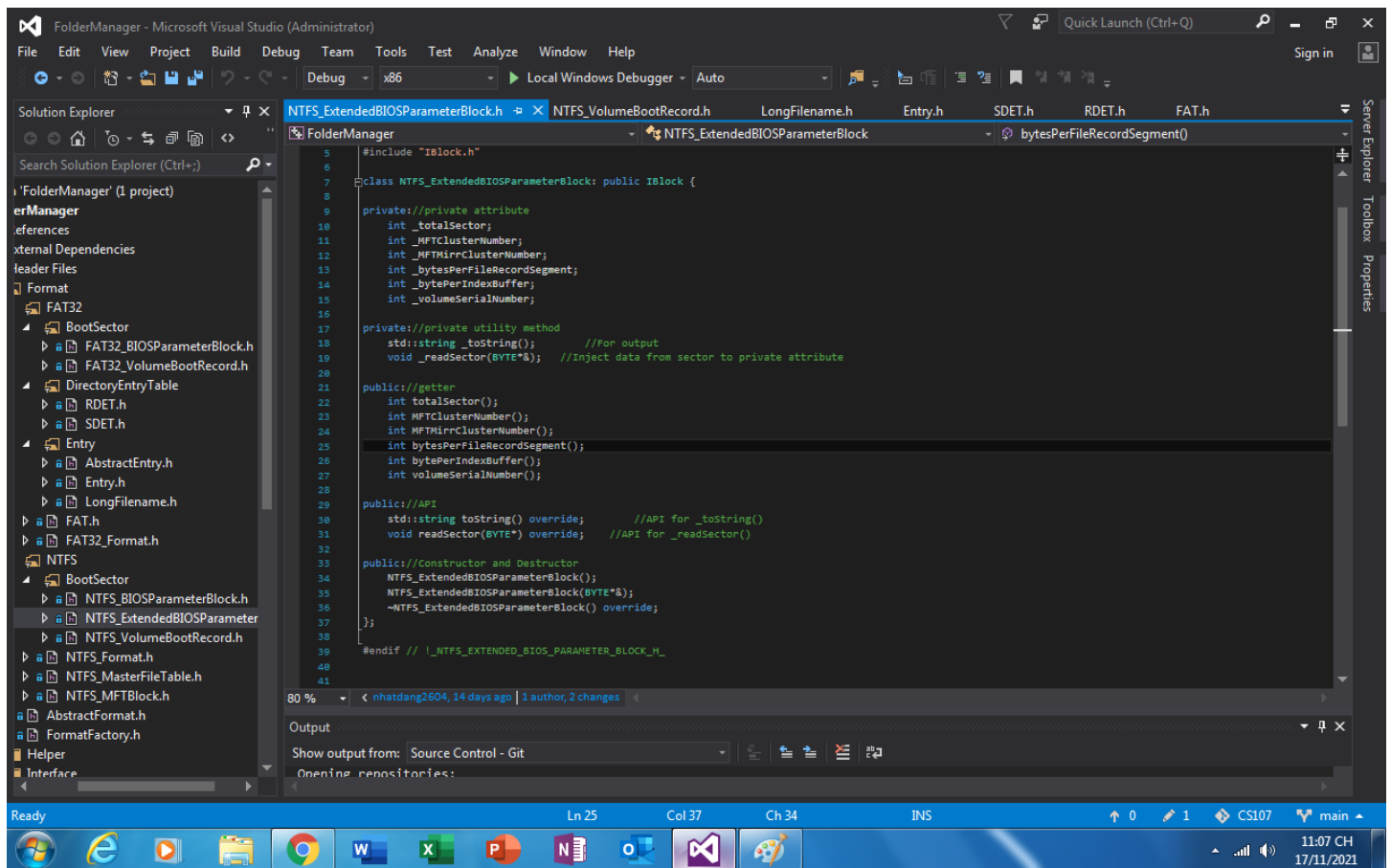
Bảng 9: Nội dung BPB



Hình 19: Thuộc tính class NTFS BIOS Parameter Block

Nội dung	Kích thước	Offset
Total sectors	8 byte	0x28
MFT Cluster number	8 byte	0x30
MFT Mirror Cluster number	8 byte	0x38
Bytes per File Record Segment	1 byte	0x40 (số có dấu, đổi ra byte = $2^{ n }$)
Bytes per Index Buffer	1 byte	0x44 (số có dấu, đổi ra byte = $2^{ n }$)
Volume serial number	8 byte	0x48

Bảng 10: Nội dung EBPB

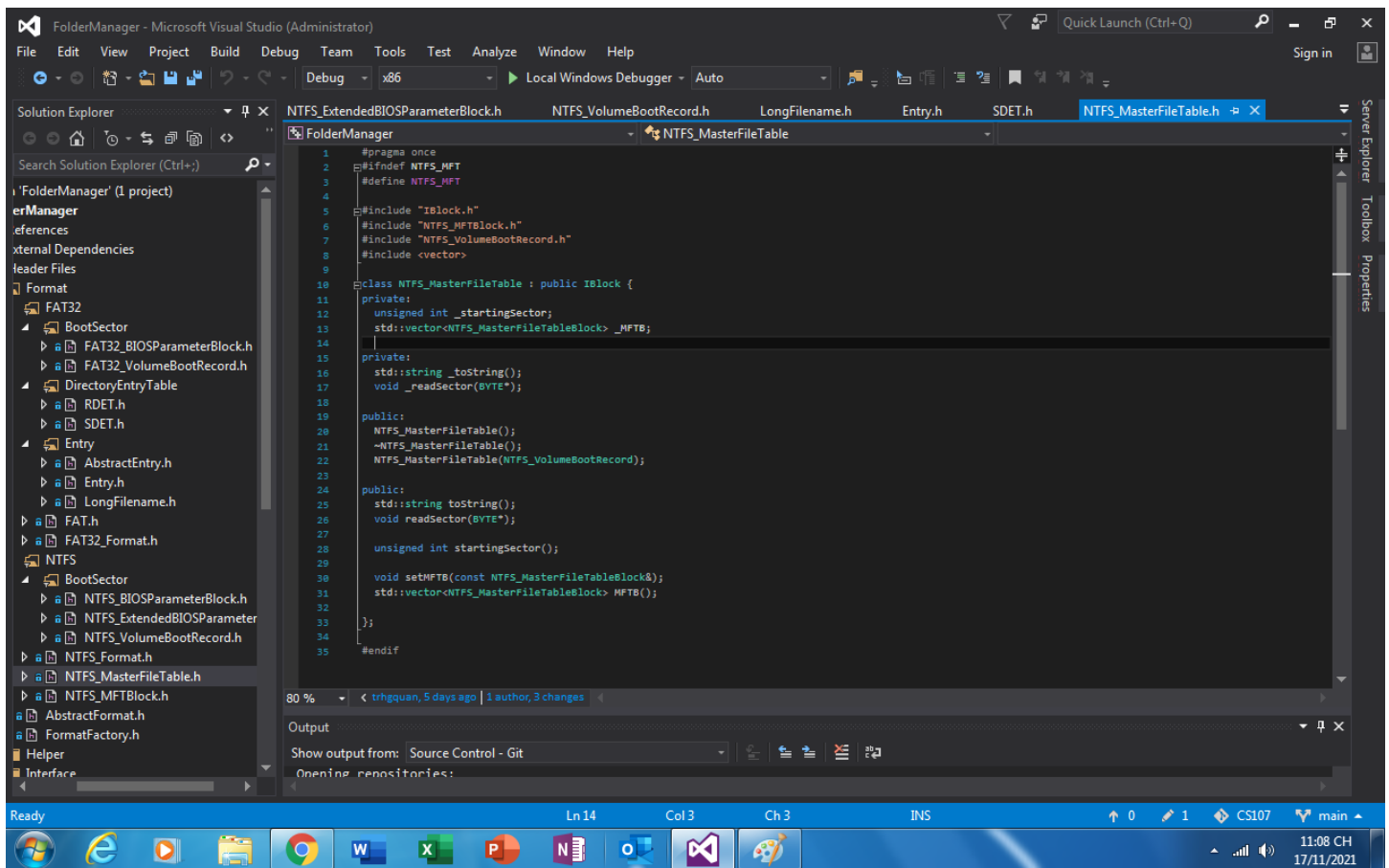


Hình 20: Thuộc tính class NTFS Extended BIOS Paramter Block

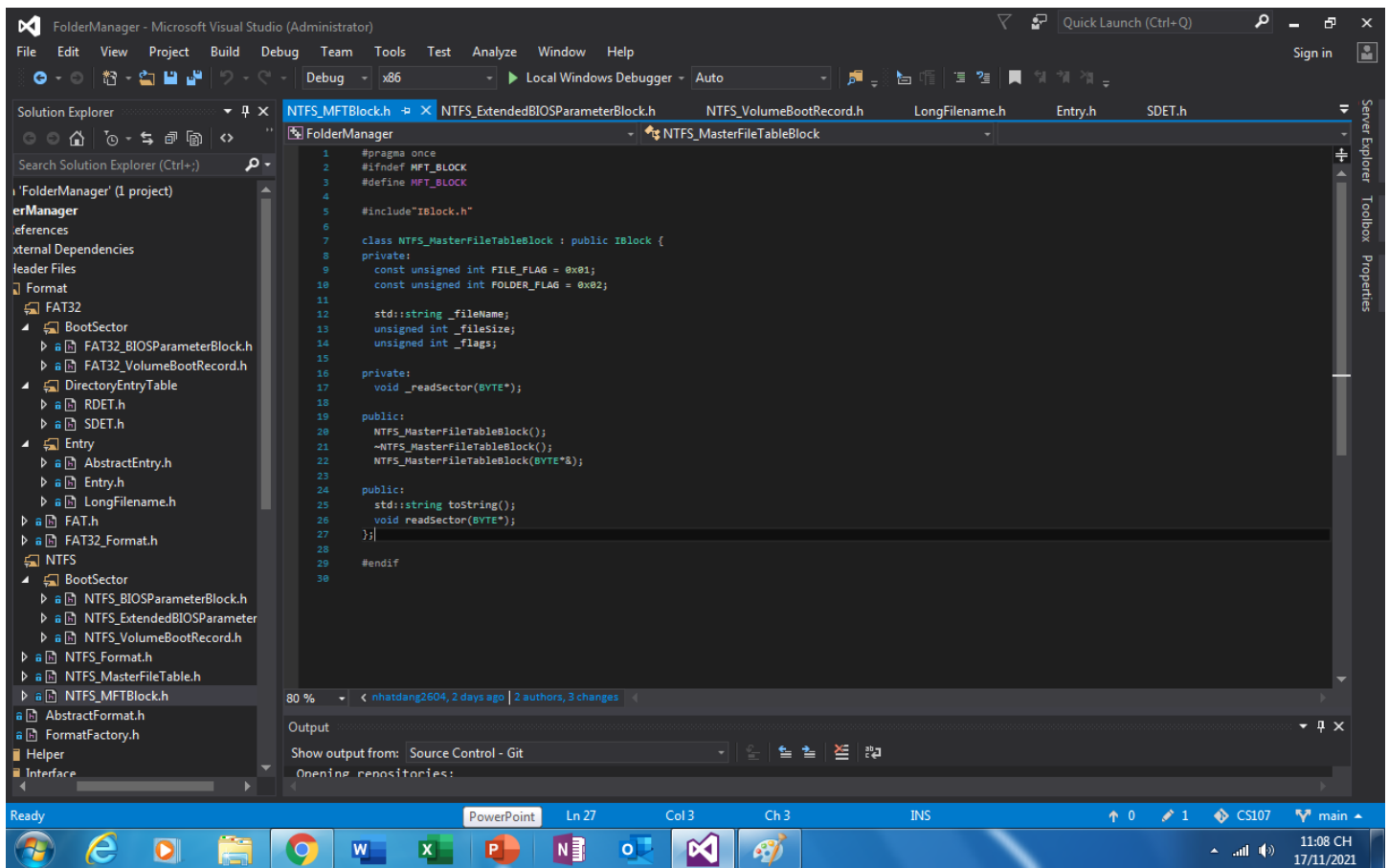
Từ EPBP, ta tìm ra được chỉ số cluster bắt đầu X . Đổi về sector bằng công thức $s = X * Sc$, ta đọc được entry đầu tiên của MFT. MFT được cài đặt trong NTFS_MasterFileTable.h. Một NTFS_MasterFileTable sẽ chứa nhiều NTFS_MFTBlock được cài đặt trong NTFS_MFTBlock.h

Nội dung	Kích thước	Offset
Signature	4 byte	0x00
Entry start	2 byte	0x14
Trạng thái	2 byte	0x16
\$STANDARD_INFORMATION size	4 byte	4 + Entry Start
\$FILENAME starting	2 byte	dòng trên + 20
Filename length	1 byte	dòng trên + 64
Filename	Filename length - byte	66 byte kể từ sau filename starting.

Bảng 11: Nội dung MFT



Hình 21: Thuộc tính class Master File Table

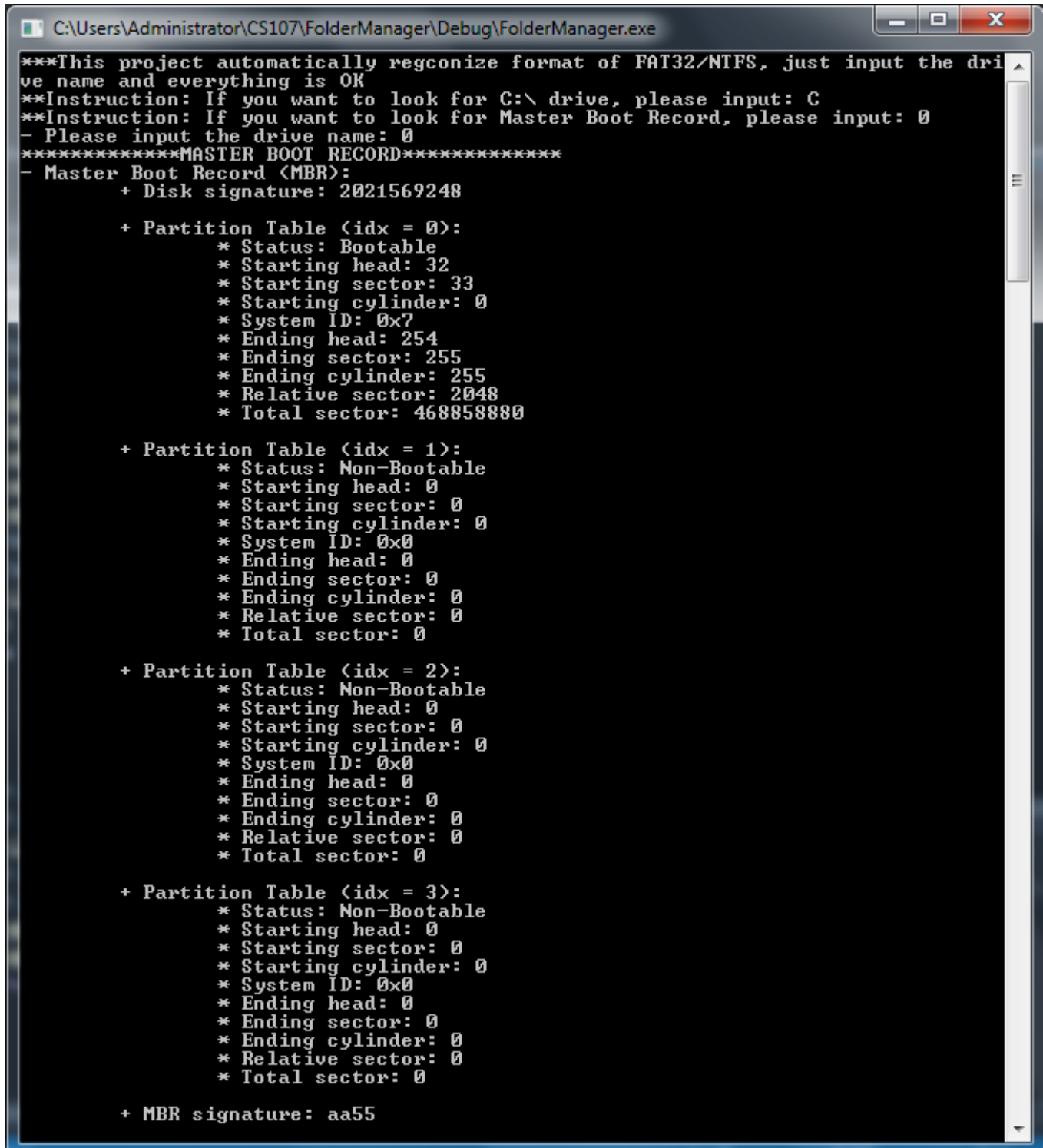


Hình 22: Thuộc tính class MFT Block

3 Demo

3.1 Kết quả chạy chương trình để test Master Boot Record

3.1.1 Thông tin trên Master Boot Record



```
C:\Users\Administrator\CS107\FolderManager\Debug\FolderManager.exe
***This project automatically recognize format of FAT32/NTFS, just input the drive name and everything is OK
**Instruction: If you want to look for C:\ drive, please input: C
**Instruction: If you want to look for Master Boot Record, please input: 0
- Please input the drive name: 0
*****MASTER BOOT RECORD*****
- Master Boot Record (MBR):
  + Disk signature: 2021569248

  + Partition Table (idx = 0):
    * Status: Bootable
    * Starting head: 32
    * Starting sector: 33
    * Starting cylinder: 0
    * System ID: 0x7
    * Ending head: 254
    * Ending sector: 255
    * Ending cylinder: 255
    * Relative sector: 2048
    * Total sector: 468858880

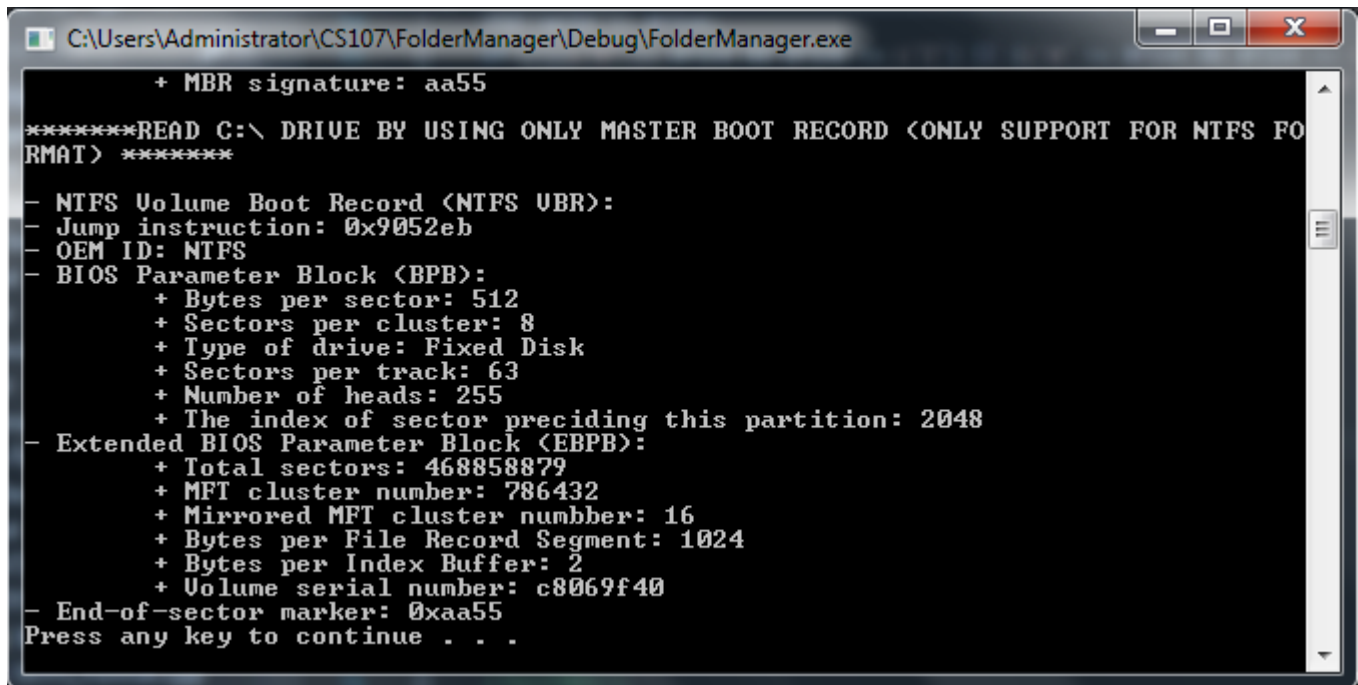
  + Partition Table (idx = 1):
    * Status: Non-Bootable
    * Starting head: 0
    * Starting sector: 0
    * Starting cylinder: 0
    * System ID: 0x0
    * Ending head: 0
    * Ending sector: 0
    * Ending cylinder: 0
    * Relative sector: 0
    * Total sector: 0

  + Partition Table (idx = 2):
    * Status: Non-Bootable
    * Starting head: 0
    * Starting sector: 0
    * Starting cylinder: 0
    * System ID: 0x0
    * Ending head: 0
    * Ending sector: 0
    * Ending cylinder: 0
    * Relative sector: 0
    * Total sector: 0

  + Partition Table (idx = 3):
    * Status: Non-Bootable
    * Starting head: 0
    * Starting sector: 0
    * Starting cylinder: 0
    * System ID: 0x0
    * Ending head: 0
    * Ending sector: 0
    * Ending cylinder: 0
    * Relative sector: 0
    * Total sector: 0

  + MBR signature: aa55
```

3.1.2 Thông tin trên bootable Boot Sector khi sử dụng thông tin từ Master Boot Record để đọc



```
C:\Users\Administrator\CS107\FolderManager\Debug\FolderManager.exe

+ MBR signature: aa55

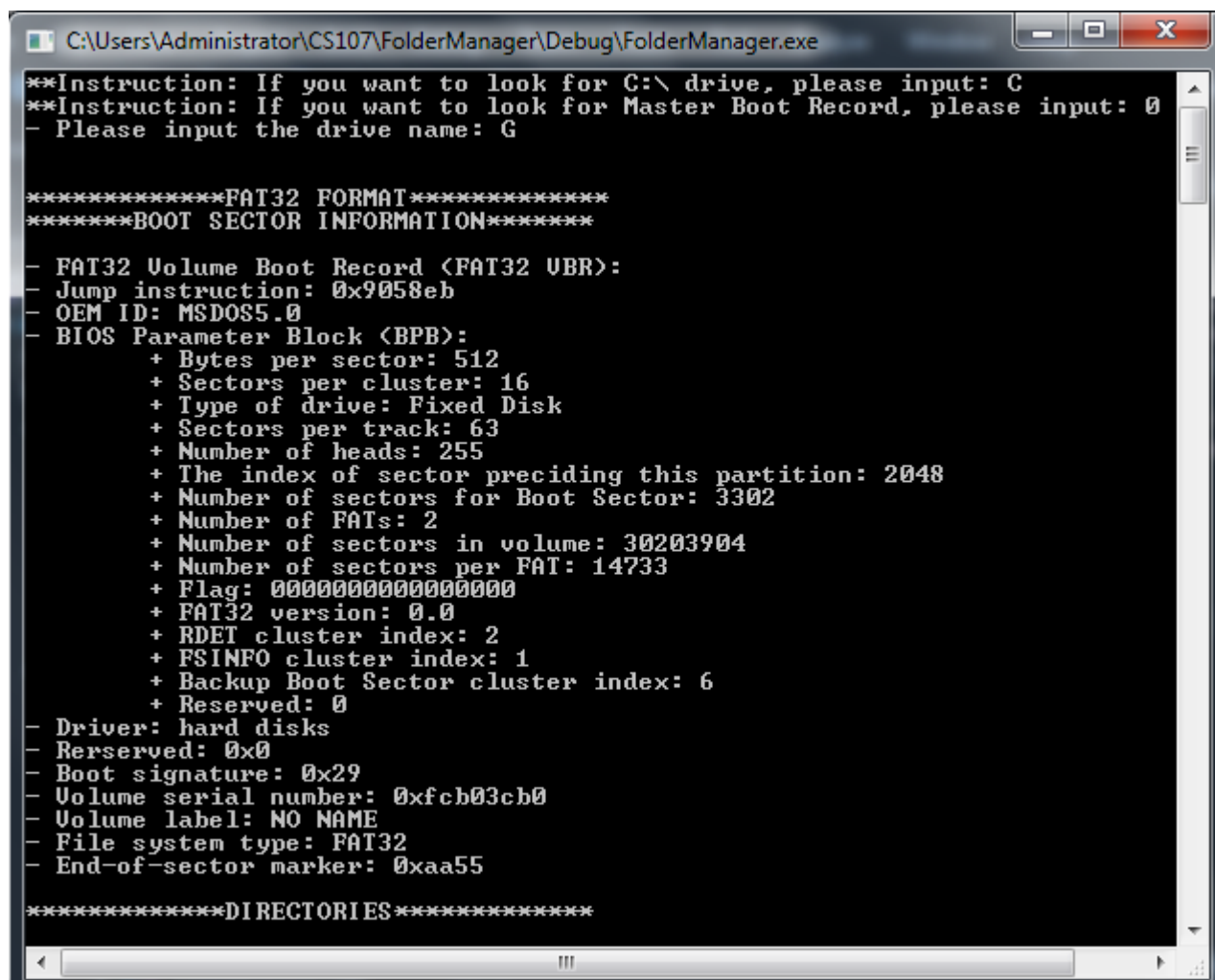
*****READ C:\ DRIVE BY USING ONLY MASTER BOOT RECORD (ONLY SUPPORT FOR NTFS FORMAT) *****

- NTFS Volume Boot Record (NTFS VBR):
- Jump instruction: 0x9052eb
- OEM ID: NTFS
- BIOS Parameter Block (BPB):
  + Bytes per sector: 512
  + Sectors per cluster: 8
  + Type of drive: Fixed Disk
  + Sectors per track: 63
  + Number of heads: 255
  + The index of sector preceding this partition: 2048
- Extended BIOS Parameter Block (EBPB):
  + Total sectors: 468858879
  + MFT cluster number: 786432
  + Mirrored MFT cluster number: 16
  + Bytes per File Record Segment: 1024
  + Bytes per Index Buffer: 2
  + Volume serial number: c8069f40
- End-of-sector marker: 0xaa55
Press any key to continue . . .
```

Hình 24: Kết quả chạy chương trình trên Master Boot Record (2)

3.2 Kết quả chạy chương trình trên USB FAT32

3.2.1 Thông tin trên Boot Sector



```
C:\Users\Administrator\CS107\FolderManager\Debug\FolderManager.exe

**Instruction: If you want to look for C:\ drive, please input: C
**Instruction: If you want to look for Master Boot Record, please input: 0
- Please input the drive name: G

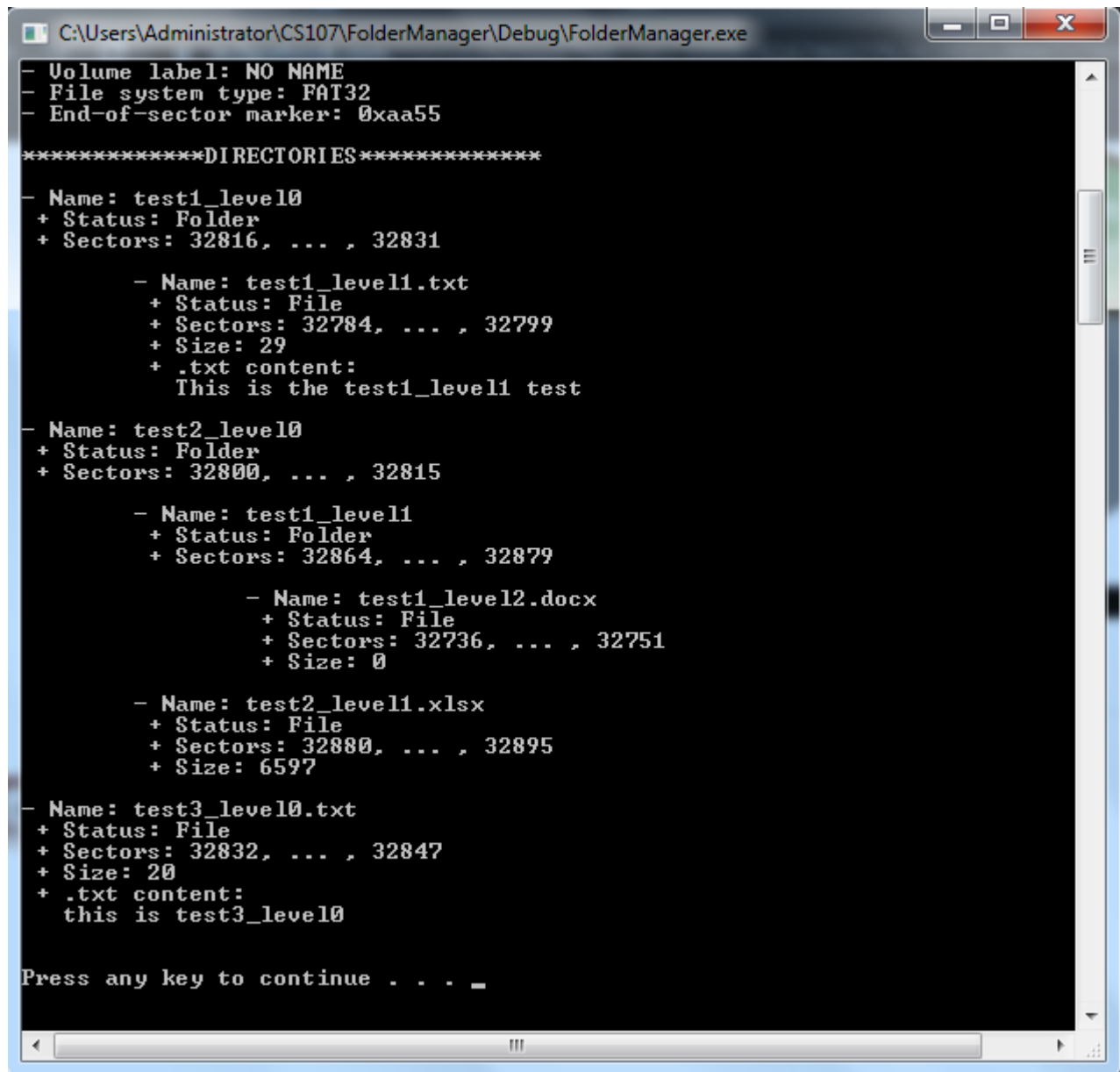
*****FAT32 FORMAT*****
*****BOOT SECTOR INFORMATION*****

- FAT32 Volume Boot Record <FAT32 UBR>:
- Jump instruction: 0x9058eb
- OEM ID: MSDOS5.0
- BIOS Parameter Block <BPB>:
  + Bytes per sector: 512
  + Sectors per cluster: 16
  + Type of drive: Fixed Disk
  + Sectors per track: 63
  + Number of heads: 255
  + The index of sector preceding this partition: 2048
  + Number of sectors for Boot Sector: 3302
  + Number of FATs: 2
  + Number of sectors in volume: 30203904
  + Number of sectors per FAT: 14733
  + Flag: 0000000000000000
  + FAT32 version: 0.0
  + RDET cluster index: 2
  + FSINFO cluster index: 1
  + Backup Boot Sector cluster index: 6
  + Reserved: 0
- Driver: hard disks
- Rerserved: 0x0
- Boot signature: 0x29
- Volume serial number: 0xfcb03cb0
- Volume label: NO NAME
- File system type: FAT32
- End-of-sector marker: 0xaa55

*****DIRECTORIES*****
```

Hình 25: Kết quả chạy chương trình trên ổ đĩa định dạng FAT32 (1)

3.2.2 Cây thư mục trên ổ đĩa



```
C:\Users\Administrator\CS107\FolderManager\Debug\FolderManager.exe
- Volume label: NO NAME
- File system type: FAT32
- End-of-sector marker: 0xaa55

*****DIRECTORIES*****

- Name: test1_level0
+ Status: Folder
+ Sectors: 32816, ... , 32831
  - Name: test1_level1.txt
  + Status: File
  + Sectors: 32784, ... , 32799
  + Size: 29
  + .txt content:
    This is the test1_level1 test

- Name: test2_level0
+ Status: Folder
+ Sectors: 32800, ... , 32815
  - Name: test1_level1
  + Status: Folder
  + Sectors: 32864, ... , 32879
    - Name: test1_level2.docx
    + Status: File
    + Sectors: 32736, ... , 32751
    + Size: 0

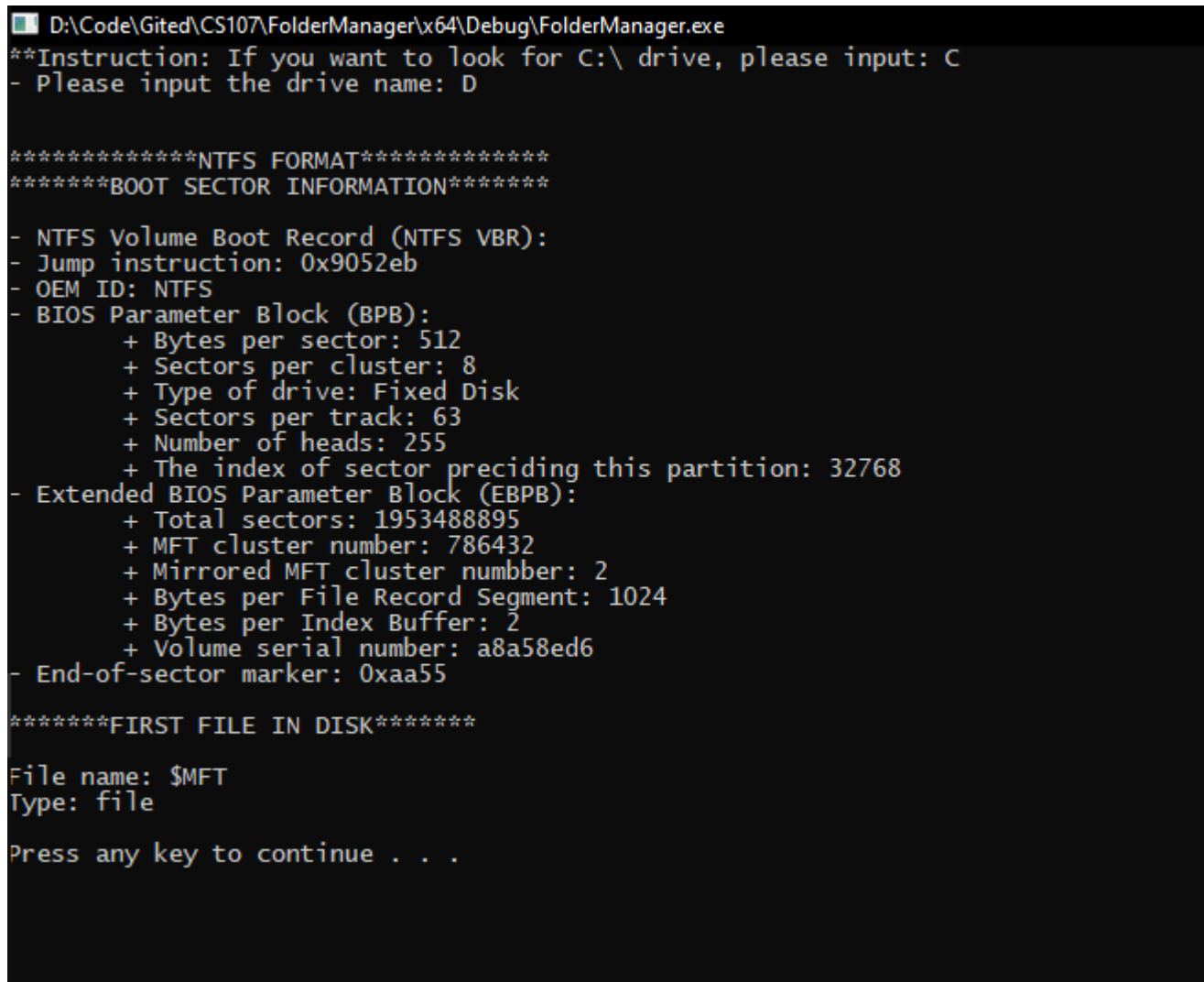
    - Name: test2_level1.xlsx
    + Status: File
    + Sectors: 32880, ... , 32895
    + Size: 6597

- Name: test3_level0.txt
+ Status: File
+ Sectors: 32832, ... , 32847
+ Size: 20
+ .txt content:
  this is test3_level0

Press any key to continue . . . _
```

Hình 26: Kết quả chạy chương trình trên ổ đĩa định dạng FAT32 (2)

3.3 Kết quả chạy chương trình trên ổ đĩa định dạng NTFS



```
D:\Code\Gited\CS107\FolderManager\x64\Debug\FolderManager.exe
**Instruction: If you want to look for C:\ drive, please input: C
- Please input the drive name: D

*****NTFS FORMAT*****
*****BOOT SECTOR INFORMATION*****

- NTFS Volume Boot Record (NTFS VBR):
- Jump instruction: 0x9052eb
- OEM ID: NTFS
- BIOS Parameter Block (BPB):
  + Bytes per sector: 512
  + Sectors per cluster: 8
  + Type of drive: Fixed Disk
  + Sectors per track: 63
  + Number of heads: 255
  + The index of sector preceding this partition: 32768
- Extended BIOS Parameter Block (EBPB):
  + Total sectors: 1953488895
  + MFT cluster number: 786432
  + Mirrored MFT cluster numbber: 2
  + Bytes per File Record Segment: 1024
  + Bytes per Index Buffer: 2
  + Volume serial number: a8a58ed6
- End-of-sector marker: 0xaa55

*****FIRST FILE IN DISK*****

File name: $MFT
Type: file

Press any key to continue . . .
```

Hình 27: Kết quả chạy chương trình trên ổ đĩa định dạng NTFS

Tài liệu

- [1] Lê Viết Long, *Bài giảng: Hệ thống tập tin FAT*
- [2] Lê Viết Long, *Bài giảng: Hệ thống tập tin*
- [3] Trần Trung Dũng, Phạm Thái Sơn (2013), *Hệ Điều Hành*
- [4] Paul Stoffregen (2005), *Understanding FAT32 Filesystems*: <https://www.pjrc.com/tech/8051/ide/fat32.html>
- [5] Minatu (2015), *Khái quát về FAT32*: <https://lazytrick.wordpress.com/2015/12/27/khai-quat-ve-fat/>
- [6] Behzad Mahjour Shafiei, Farshid Iranmanesh, Fariborz Iranmanesh, Bardsir Branch (2012), *Review NTFS Basics*: <http://www.ajbasweb.com/old/ajbas/2012/July/325-338.pdf>
- [7] Richard Russon, Yuval Fledel, *NTFS Documentation*: <https://dubeyko.com/development/FileSystems/NTFS/ntfsdoc.pdf>