KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
227 Nguyễn Văn Cừ, Phường 4, Quận 5, TP.HCM
Điện Thoại: (08) 38.354.266 - Fax:(08) 38.350.096

ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐẠI HỌC QUỐC GIA TP HCM

KHOA CÔNG NGHỆ THÔNG TIN

ĐỒ ÁN
HỆ ĐIỀU HÀNH

Quản lý hệ thống tập tin trên Windows

Sinh viên thực hiện:
19120485 – Nguyễn Phạm Quang Dũng
19120489 – Lưu Trường Dương
19120497 – Bùi Trường Giang
19120661 – Lê Mai Nguyên Thảo
18120306 – Lê Thọ Đạt
GV phụ trách: Ths. Lê Viết Long

HỌC KỲ I – NĂM HỌC 2021-2022

MỤC LỤC

A. Thông tin nhóm và phân công công việc

THÔNG TIN NHÓM VÀ PHÂN CÔNG CÔNG VIỆC				
Số lượng		5 thành viên		
STT	**MSSV**	**Họ tên**	**Công việc, nghiên cứu**	**Mức độ hoàn thành**
1	19120485	Nguyễn Phạm Quang Dũng	- Đọc thông tin MFT của NTFS. - Render cây thư mục NTFS.	100%
2	19120489	Lưu Trường Dương	- Tìm hiểu FAT32, thiết kế các hàm chức năng để có thể đọc được bảng sector của FAT32 và NTFS.	100%
3	19120497	Bùi Trường Giang	- Tìm hiểu FAT32, NTFS và các hàm hệ thống, đọc Boostector. - Sửa lỗi.	100%
4	19120661	Lê Mai Nguyên Thảo	- Đọc thông tin RDET, SDET của FAT32 - Render cây thư mục FAT32	100%
5	18120306	Lê Thọ Đạt	- Phân cấp cây thư mục NTFS. Định dạng file và loại chương trình mở - Viết báo cáo	100%

Mức độ hoàn thành toàn bộ Project	100%

B. Nội dung

I. Các bước thực hiện

1. Các hàm hỗ trợ:

- Đây là các hàm dùng trong toàn bộ chương trình để đơn giản hóa các quá trình đi lặp lại như:
  - Chuyển bảng sector thành một vector 2 chiều dạng string

```
vector<vector<string>> to_vector(BYTE sector[512]) {
    vector<vector<string>> result;
    int i = 0;

    for (int i = 0; i < 32; i++) {
        vector<string> temp;
        for (int j = 0; j < 16; j++) {
            stringstream ss;
            ss << setfill('0') << setw(2) << uppercase << hex << static_cast<unsigned>(sector[i * 16 + j]);
            temp.push_back(ss.str());
        }
        result.push_back(temp);
    }
    return result;
}
```

  - Đọc bộ string dạng hex từ bảng sector:

```
string to_hexstr(vector<vector<string>> sector, int col, int row, int num_byte, bool flag) {
    string result;
    if (flag)
        for (int i = num_byte - 1; i >= 0; i--)
            result += sector[row][col + i];
    else
        for (int i = 0; i < num_byte; i++)
            result += sector[row][col + i];

    return result;
}
```

  - Chuyển đổi string dạng hex sang dạng đọc được:

```
string hexstr_tostr(string hex) {
    string ascii = "";
    for (size_t i = 0; i < hex.length(); i += 2)
    {
        string part = hex.substr(i, 2);
        char ch = stoul(part, nullptr, 16);
        ascii += ch;
    }
    return ascii;
}
```

- Chuyển từ string dạng hex sang int:

```
int hexstr_to_int(string hexstr)
{
    int result;
    char* offset;
    string real_hex = "0x" + hexstr;
    result = strtol(real_hex.c_str(), &offset, 0);
    return result;
}
```

- Chuyển số int sang dạng nhị phân:

```
string to_binstr(int number)
{
    string res;
    while (number != 0)
    {
        res = (number % 2 == 0 ? "0" : "1") + res;
        number /= 2;
    }
    if (res.length() < 8)
    {
        string sub;
        for (int i = 7 - res.length(); i >= 0; i--)
        {
            sub += '0';
        }
        res = sub + res;
    }
    return res;
}
```

- Nối 2 mảng 2 chiều dạng string:

```
vector<vector<string>> combine_table(vector<vector<string>> v1, vector<vector<string>> v2) {
    /*int begin = v1.size();*/
    for (int i = 0; i < v2.size(); i++) {
        v1.push_back(v2[i]);
    }
    v2.clear();
    return v1;
}
```

- Kiểm tra kết thúc:

```
bool is_end(vector<vector<string>> vec) {
    for (int i = 0; i < vec[0].size(); i++) {
        if (vec[vec.size() - 1][i] != "00") return false;
    }
    return true;
}
```

- Đọc nội dung entry dùng phương pháp đệ quy:

```
vector<vector<string>> find_table(HANDLE device, int readPont)
{
    DWORD bytesRead;
    BYTE sector[512];

    long a = 0x0;

    vector<vector<string>> result;
    SetFilePointer(device, readPont, &a, FILE_BEGIN);//Set a Point to Read

    if (!ReadFile(device, sector, 512, &bytesRead, NULL))
    {
        printf("Find sector false at: %u\n", GetLastError());
        return result;
    }

    result = to_vector(sector);

    if (is_end(result)) {
        return result;
    }

    vector<vector<string>> next = find_table(device, readPont + 512);
    return combine_table(result, next);
};
```

- In nội dung dạng hex của entry ra stdout:

```
void print_table(vector<vector<string>> vec) {
    for (int i = 0; i < vec.size(); i++) {
        for (int j = 0; j < vec[i].size(); j++) {
            cout << vec[i][j] << " ";
        }
        cout << endl;
    }
    return;
}
```

- In tab (phân cấp cây thư mục):

```
void print_Tab(int k)
{
    if (k == 0)
        return;
    if (k >= 1)
    {
        for (int i = 0; i < k; i++)
        {
            cout << "\t";
        }
    }
}
```

- Kiểm tra loại phần mềm đọc file:

```
string getOpener(string ext)
{
    string opener = "";
    if (ext == "txt" || ext == "TXT")
        return "";
    if (ext == "doc" || ext == "DOC" || ext == "docx" || ext == "DOCX")
        opener = "Microsoft Word";
    if (ext == "xls" || ext == "XLS" || ext == "xlsx" || ext == "XLSX")
        opener = "Microsoft Excel";
    if (ext == "pptx" || ext == "PPTX")
        opener = "Microsoft Powerpoint";
    if (ext == "pdf" || ext == "PDF")
        opener = "Adobe Reader XI";
    if (ext == "mp3" || ext == "MP3" || ext == "mp4" || ext == "MP4" || ext == "wmv" || ext == "WMV" || ext == "wma" || ext == "WMA")
        opener = "Windows Media Player";
    if (ext == "jpg" || ext == "JPG" || ext == "jpeg" || ext == "JPEG" || ext == "png" || ext == "PNG")
        opener = "Photos";
    if (ext == "accdb" || ext == "ACCDB")
        opener = "Microsoft Access";
    if (ext == "sql" || ext == "SQL")
        opener = "SQL Server or MySQL";
    if (ext == "pub" || ext == "PUB")
        opener = "Microsoft Publisher";
    if (ext == "rar" || ext == "RAR" || ext == "rar4" || ext == "RAR4" || ext == "zip" || ext == "ZIP")
        opener = "WinRAR";
    if (ext == "rtf" || ext == "RTF")
        opener = "WordPad";
    if (opener.length() == 0)
    {
        return "Unknown extension";
    }
    return "Open " + ext + " file by: " + opener;
}
```

- Tách file name và extension:

```
vector<string> split_File_Name(string file_name, char ch)
{
    vector<string> res;
    string fea;
    stringstream line;
    if (file_name != "")
    {
        line << file_name;
        while (getline(line, fea, ch))
            res.push_back(fea);
    }
    return res;
}
```

- Lấy file extension:

```
string get_ext_file(string file_name)
{
    vector<string> name_fea = split_File_Name(file_name, '.');
    string ext = name_fea[1];
    string ext1;
    for (int i = 0; i < ext.size(); i++)
    {
        if (ext[i] == 'a' or ext[i] == 'b' or ext[i] == 'c' or ext[i] == 'd' or ext[i] == 'e' or ext[i] == 'f' or ext[i] == 'g'
            or ext[i] == 'h' or ext[i] == 'i' or ext[i] == 'j' or ext[i] == 'k' or ext[i] == 'l' or ext[i] == 'm' or ext[i] == 'n'
            or ext[i] == 'o' or ext[i] == 'p' or ext[i] == 'q' or ext[i] == 'r' or ext[i] == 's' or ext[i] == 't' or ext[i] == 'u'
            or ext[i] == 'v' or ext[i] == 'w' or ext[i] == 'x' or ext[i] == 'y' or ext[i] == 'z'
            or ext[i] == 'A' or ext[i] == 'B' or ext[i] == 'C' or ext[i] == 'D' or ext[i] == 'E' or ext[i] == 'F' or ext[i] == 'G'
            or ext[i] == 'H' or ext[i] == 'I' or ext[i] == 'J' or ext[i] == 'K' or ext[i] == 'L' or ext[i] == 'M' or ext[i] == 'N'
            or ext[i] == 'O' or ext[i] == 'P' or ext[i] == 'Q' or ext[i] == 'R' or ext[i] == 'S' or ext[i] == 'T' or ext[i] == 'U'
            or ext[i] == 'V' or ext[i] == 'W' or ext[i] == 'X' or ext[i] == 'Y' or ext[i] == 'Z')
        {
            ext1 += ext[i];
        }
    }
    return ext1;
}
```

2. Khởi động chương trình

- Đọc thông tin trên ổ đĩa, chuyển sang chuỗi string và kiểm tra loại ổ, từ đó tạo các instance của các class tương ứng với ổ đĩa đó (FAT32, NTFS) và đọc thông tin, in ra màn hình

```
int main(int argc, char** argv)
{
    vector<vector<string>> sector;
    HANDLE disk = NULL;
    int flag = ReadSector(disk, sector);

    if (!flag)
        return 0;

    // chuyen nhap vi tri vao, so byte de lay tu vec ra chuoi string hex
    string temp = to_hexstr(sector, 2, 5, 5, 0);

    // chuyen chuoi string hex thanh int
    string result = hexstr_tostr(temp);

    if (result == "FAT32")
    {
        FAT32 drive(disk, sector);
        cout << "--------------------|DRIVE INFO|--------------------\n";
        drive.readBoot_Sector();
        cout << "\n----------------------|RDET|----------------------\n";

        drive.read_RDet(find_table(disk, drive.first_Sector_Of_Data() * 512), 0);
    }
    else
    {
        NTFS drive(disk, sector);
        cout << "--------------------|DRIVE INFO|--------------------\n";
        drive.readBoot_Sector();
        cout << "\n----------------------|MFT|----------------------\n";
        drive.read_MFT(find_table(disk, drive.get_first_sector_MFT() * 512), drive);
        drive.findSubDirectory();
    }

    return 1;
}
```

- Đối với FAT32, thực hiện đọc thông tin từ bảng RDET với vị trí sector bắt đầu của data đã tính trước đó.
- Đối với NTFS, thực hiện đọc bảng MFT với sector đầu tiên của MFT, sau đó thực hiên phân cấp thư mục từ danh sách các entry đã đọc.

3. FAT

- Class FAT32 dùng để lưu trữ các thông tin và method cần thiết.
    - Hàm readBoot_Sector dùng để đọc nội dung các thông số như: số bytes trên một sector, số sector trên một cluster, số sector phần bootsector, số bảng FAT, số sector trên một bảng FAT, kích thước của volume, sector đầu tiên của bảng FAT, RDET và vùng data.

```
void FAT32::readBoot_Sector()
{
    cout << "- Type of File System: FAT32" << endl;

    bytes_per_sector = hexstr_to_int(to_hexstr(BootSector, 11, 0, 2, 1));
    cout << "- The number of bytes per sector: " << bytes_per_sector << endl;

    sectors_per_cluster = hexstr_to_int(to_hexstr(BootSector, 13, 0, 1, 1));
    cout << "- The number of sectors per cluster: " << sectors_per_cluster << endl;

    sectors_of_boot = hexstr_to_int(to_hexstr(BootSector, 14, 0, 2, 1));
    cout << "- The number of bytes in Boot Sector: " << sectors_of_boot << endl;

    numbers_of_fats = hexstr_to_int(to_hexstr(BootSector, 0, 1, 1, 1));
    cout << "- The number of FAT table: " << numbers_of_fats << endl;

    cout << "- Size of Volumn: " << hexstr_to_int(to_hexstr(BootSector, 0, 2, 4, 1)) << endl;

    sectors_per_fat = hexstr_to_int(to_hexstr(BootSector, 4, 2, 4, 1));
    cout << "- Size of FAT table: " << sectors_per_fat << endl;

    cout << "- First sector of FAT table: " << sectors_of_boot << endl;

    cout << "- First sector of RDET: " << sectors_of_boot + numbers_of_fats * sectors_per_fat << endl;

    cout << "- First sector of Data: " << sectors_of_boot + numbers_of_fats * sectors_per_fat << endl;
}
```

- Hàm first_Sector_Of_data dùng để tìm sector đầu tiên của vùng data bằng cách cộng sector vùng bootsector, số bảng FAT và số sector trên 1 bảng FAT.

```
int FAT32::first_Sector_Of_Data()
{
    return sectors_of_boot + numbers_of_fats * sectors_per_fat;
}
```

- Hàm split_Entry thực hiện lọc, đọc các entry và loại bỏ các entry không cần thiết:

```
if (table[4][0] != "00")    // Folder bi rong
{
    for (int i = 0; i < table.size();)
    {
        string x = to_hexstr(table, 11, i, 1, 1);
        if (x != "0F")      // La entry chinh
        {
            vector<vector<string>> temp;
            for (int j = i; j < i + 2; j++)
            {
                temp.push_back(table[j]);
            }
            res.push_back(temp);
            i += 2;
        }
        else
        {
            int k = i + 2;
            int count = 1;
            while (to_hexstr(table, 11, k, 1, 1) == "0F")
            {
                count = count + 1;
                k = k + 2;
            }
            vector<vector<string>> temp;
            for (int j = i; j < i + 2 * (count + 1); j++)
            {
                temp.push_back(table[j]);
            }
            res.push_back(temp);
            i += 2 * (count + 1);
        }
    }
}
```

Kiểm tra folder bị rỗng. Kiểm tra nếu entry là entry chính, đẩy vào bảng kết quả. Nếu là entry phụ, thực hiện đọc tiếp nội dung cho tới khi tìm được entry chính, sau đó đẩy toàn bộ entry tìm được vào bảng kết quả.

```
while (res[0][0][0] == "2E")
{
    res.erase(res.begin());
}
```

Loại bỏ các bản ghi thư mục gốc

```
for (int i = 0; i < res.size();)
{
    if (to_hexstr(res[i], 11, 0, 1, 1) != "0F")
    {
        if (to_hexstr(res[i], 11, 0, 1, 1) != "20" && to_hexstr(res[i], 11, 0, 1, 1) != "10")
        {
            res.erase(res.begin() + i);
        }
        else i += 1;
    }
    else
    {
        if (to_hexstr(res[i], 11, res[i].size() - 2, 1, 1) != "20" && to_hexstr(res[i], 11, res[i].size() - 2, 1, 1) != "10")
        {
            res.erase(res.begin() + i);
        }
        else i += 1;
    }
}
```

Bỏ các entry không phải là file/folder

- Hàm check_Entry: Kiểm tra hợp lệ của entry:

```
int FAT32::check_Entry(vector<vector<string>> entry)
{
    string x;
    if (entry[0][11] != "0F")
    {
        x = to_binstr(hexstr_to_int(entry[0][11]));
    }
    else
    {
        x = to_binstr(hexstr_to_int(entry[entry.size() - 2][11]));
    }

    int cnt = 0;
    for (int i = 0; i < x.length(); i++)
    {
        if (x[i] == '1')
        {
            cnt = cnt + 1;
        }
    }

    if (cnt == 1)
        return 1;
    return 0;
}
```

- Hàm check_Character: Kiểm tra kí tự trong bảng mã ascii (phục vụ cho quá trình in tên file)

```
int FAT32::check_Character(string x)
{
    if (x[0] >= '2' && x[0] <= '7')
        return 1;
    return 0;
}
```

```
int FAT32::check_Entry(vector<vector<string>> entry)
{
    string x;
    if (entry[0][11] != "0F")
    {
        x = to_binstr(hexstr_to_int(entry[0][11]));
    }
    else
    {
        x = to_binstr(hexstr_to_int(entry[entry.size() - 2][11]));
    }

    int cnt = 0;
    for (int i = 0; i < x.length(); i++)
    {
        if (x[i] == '1')
        {
            cnt = cnt + 1;
        }
    }

    if (cnt == 1)
        return 1;
    return 0;
}
```

- Hàm get_First_Cluster: lấy thông tin cluster đầu tiên của entry:

```
int FAT32::get_First_Cluster(vector<vector<string>> entry)
{
    if (entry[0][11] != "0F")
    {
        return hexstr_to_int(to_hexstr(entry, 4, 1, 2, 1) + to_hexstr(entry, 10, 1, 2, 1));
    }
    else
    {
        return hexstr_to_int(to_hexstr(entry, 4, entry.size() - 1, 2, 1) + to_hexstr(entry, 10, entry.size() - 1, 2, 1));
    }
}
```

- Hàm get_First_Sector: từ thông tin cluster đầu tiên đã có trước đó, thực hiện tìm vị trí sector đầu tiên của entry trong bảng Data:

```
int FAT32::get_First_Sector(vector<vector<string>> entry)
{
    return first_Sector_Of_Data() + sectors_per_cluster * (get_First_Cluster(entry) - 2);
}
```

- Hàm find_Attribute: đọc thông tin, thuộc tính của entry (file/folder) tương ứng theo phương pháp bit bật (read only, hidden, system, directory,...)

```
for (i = 0; i < entry.size(); i++)
{
    if (to_hexstr(entry, 11, i, 1, 1) != "0F" && to_hexstr(entry, 11, i, 1, 1) != "00")
        res_hex = to_hexstr(entry, 11, i, 1, 1);
}
res_bin = to_binstr(hexstr_to_int(res_hex));

i = res_bin.length() - 1;
while (res_bin[i] != '1')
{
    i--;
}
```

Lấy vùng thông tin thuộc tính

```
if (check_Entry(entry))
{
    switch (i)
    {
    case 7:
        return "Read Only";
        break;
    case 6:
        return "Hidden";
        break;
    case 5:
        return "System";
        break;
    case 4:
        return "Volumn Label
        break;
    case 3:
        return "Directory";
        break;
    case 2:
        return "Archive";
        break;
    case 1:
        return "Device";
        break;
    case 0:
        return "Unused";
        break;
    }
}
else
{
    return "Error!";
```

Nếu entry hợp lệ thì thực hiện tìm thuộc tính của file/folder

- Hàm find_name: Đọc tên của file/folder từ entry (các ký tự không hợp lệ bị loại bỏ)

```
string FAT32::find_Name(vector<vector<string>> entry)
{
    string temp = to_hexstr(entry, 11, 0, 1, 1);
    string res = "";
    if (temp != "00" && temp != "0F")
    {
        if (temp == "10")
        {
            return hexstr_tostr(to_hexstr(entry, 0, 0, 11, 0));
        }

        res = hexstr_tostr(to_hexstr(entry, 0, 0, 8, 0));
        while (res[res.length() - 1] == ' ')
        {
            res.erase(res.begin() + res.length() - 1);
        }
        res += ".";
        res += hexstr_tostr(to_hexstr(entry, 8, 0, 3, 0));
        return res;
    }

    if (temp != "00" && temp == "0F")
    {
        int i = entry.size() - 4;
        while (i >= 0)
        {
            for (int j = 1; j < 16; j++)
            {
                if (j != 11 && j != 13 && entry[i][j] != "00" && check_Character(entry[i][j]) == 1)
                    res += entry[i][j];
            }
            for (int j = 0; j < 16; j++)
            {
                if (j != 11 && entry[i + 1][j] != "00" && check_Character(entry[i + 1][j]) == 1)
                    res += entry[i + 1][j];
            }
            i = i - 2;
        }
        return hexstr_tostr(res);
    }
}
```

- Hàm is_Folder_Empty: kiểm tra folder rỗng

```
int FAT32::is_Folder_Empty(vector<vector<string>> entry)
{
    vector<vector<string>> sdet = find_table(device, get_First_Sector(entry) * 512);

    if (sdet[4][0] == "00")
        return 1;
    return 0;
}
```

- Hàm read_Data: Đọc nội dung của file nếu là file txt

```
string FAT32::read_Data(int readPont, int level)
{
    string res;
    vector<vector<string>> data = find_table(device, readPont);

    for (int i = 0; i < data.size(); i++)
    {
        for (int j = 0; j < 16; j++)
        {
            res = res + data[i][j];
        }
    }

    return res;
}
```

- Hàm read_File: đọc thông tin, thông số, nội dung của file

```
print_Tab(level);
cout << "File name: " << find_Name(entry) << endl;
print_Tab(level);
int size_of_file = hexstr_to_int(to_hexstr(entry, 12, entry.size() - 1, 4, 1));
cout << "Size: " << size_of_file << endl;
print_Tab(level);
cout << "First cluster: " << get_First_Cluster(entry);

cout << "\n ";
print_Tab(level);
cout << "--> Cluster used: ";
vector<int> cluster_used;
```

Các thông tin cơ bản (tên, kích cỡ, cluster đầu tiên)

```
cout << "--> Cluster used: ";
vector<int> cluster_used;
int i = 4 * get_First_Cluster(entry);           //chi so cot trong bang fat
cluster_used.push_back(get_First_Cluster(entry));
while (1)
{
    int k = i;
    int j = 0;                          //chi so dong trong bang fat
    for (j;;)
    {
        if (i >= 16 * (j + 1))
            j++;
        else break;
    }
    while (k >= 16)                     //chi so cot luon < 16
    {
        k -= 16;
    }
    string temp = to_hexstr(FAT, k, j, 1, 1);
    if (temp != "FF" && temp != "0F" && temp != "F8")
    {
        cluster_used.push_back(hexstr_to_int(temp));
        i += 4;
    }
    else break;
}
for (int j = 0; j < cluster_used.size(); j++)
{
    cout << cluster_used[j];
    if (j < cluster_used.size() - 1)
    {
        cout << ", ";
    }
}
```

Các cluster sử dụng

```
cout << "\n ";
print_Tab(level);
cout << "--> Sector used: ";
vector<int> sector_used;
int cur_sector = get_First_Sector(entry);
for (int j = 0; j < cluster_used.size(); j++)
{
    for (int k = 0; k < sectors_per_cluster; k++)
    {
        sector_used.push_back(cur_sector);
        cur_sector = cur_sector + 1;
    }
}
for (int j = 0; j < sector_used.size(); j++)
{
    cout << sector_used[j];
    if (j < sector_used.size() - 1)
    {
        cout << ", ";
    }
}
```

In ra các sector sử dụng

```
vector<string> name_fea = split_File_Name(find_Name(entry), '.');
string ext = name_fea[1];
if (ext == "txt" || ext == "TXT")
{
    if (size_of_file != 0)
    {
        cout << "\n ";
        print_Tab(level);
        cout << "--> Read " << name_fea[1] << " file:\n";
        string res = hexstr_tostr(read_Data(sector_used[0] * 512, level));
        cout << res << endl;
    }
    else
    {
        cout << "\n";
        print_Tab(level);
        cout << "File " << name_fea[1] << " is empty!\n";
    }
}
```

Xem nội dung file nếu là TXT

```
else
{
    cout << "\n";
    print_Tab(level);
    cout << "Open " << ext << " file by ";
    if (ext == "doc" || ext == "DOC" || ext == "docx" || ext == "DOCX")
        cout << "Microsoft Word" << endl;
    if (ext == "xls" || ext == "XLS" || ext == "xlsx" || ext == "XLSX")
        cout << "Microsoft Excel" << endl;
    if (ext == "pptx" || ext == "PPTX")
        cout << "Microsoft Powerpoint" << endl;
    if (ext == "pdf" || ext == "PDF")
        cout << "Adobe Reader XI" << endl;
    if (ext == "mp3" || ext == "MP3" || ext == "mp4" || ext == "MP4" || ext == "wmv" || ext == "WMV" || ext == "wma" || ext == "WMA")
        cout << "Windows Media Player" << endl;
    if (ext == "jpg" || ext == "JPG" || ext == "jpeg" || ext == "JPEG" || ext == "png" || ext == "PNG")
        cout << "Photos" << endl;
    if (ext == "accdb" || ext == "ACCDB")
        cout << "Microsoft Access" << endl;
    if (ext == "sql" || ext == "SQL")
        cout << "SQL Server or MySQL" << endl;
    if (ext == "pub" || ext == "PUB")
        cout << "Microsoft Publisher" << endl;
    if (ext == "rar" || ext == "RAR" || ext == "rar4" || ext == "RAR4" || ext == "zip" || ext == "ZIP")
        cout << "WinRAR" << endl;
    if (ext == "rtf" || ext == "RTF")
        cout << "WordPad" << endl;
}
```

Nếu không phải file txt thì tìm loại chương trình tương thích.

- Hàm read_Folder: tương tư, ta có hàm để đọc thông tin folder

```
void FAT32::read_Folder(vector<vector<string>> entry, int level)
{
    print_Tab(level);
    cout << "Folder name: " << find_Name(entry) << endl;
    print_Tab(level);
    cout << "First cluster: " << get_First_Cluster(entry);

    cout << "\n ";
```

Tên folder, cluster đầu tiên

```
cout << "--> Cluster used: ";
vector<int> cluster_used;
int i = 4 * get_First_Cluster(entry);            //chi so cot trong bang fat
cluster_used.push_back(get_First_Cluster(entry));
while (1)
{
    int k = i;
    int j = 0;                          //chi so dong trong bang fat
    for (j;;)
    {
        if (i >= 16 * (j + 1))
            j++;
        else break;
    }
    while (k >= 16)                     //chi so cot luon < 16
    {
        k -= 16;
    }
    string temp = to_hexstr(FAT, k, j, 1, 1);
    if (temp != "FF" && temp != "0F")
    {
        cluster_used.push_back(hexstr_to_int(temp));
        i += 4;
    }
    else break;
}
for (int j = 0; j < cluster_used.size(); j++)
{
    cout << cluster_used[j];
    if (j < cluster_used.size() - 1)
    {
        cout << ", ";
    }
}
```

Tương tự file, ta có cluster sử dụng

```
cout << "\n ";
print_Tab(level);
cout << "--> Sector used: ";
vector<int> sector_used;
int cur_sector = get_First_Sector(entry);
for (int j = 0; j < cluster_used.size(); j++)
{
    for (int k = 0; k < sectors_per_cluster; k++)
    {
        sector_used.push_back(cur_sector);
        cur_sector = cur_sector + 1;
    }
}
for (int j = 0; j < sector_used.size(); j++)
{
    cout << sector_used[j];
    if (j < sector_used.size() - 1)
    {
        cout << ", ";
    }
}
cout << endl;
```

Các sector sử dụng

- Hàm read_RDET: hàm chính dùng để đọc thông tin từ bảng RDET

```
void FAT32::read_RDet(vector<vector<string>> table, int level)
{
    FAT = find_table(device, sectors_of_boot * 512);
    vector<vector<vector<string>>> table_entry = split_Entry(table);

    if (table_entry[0][0][0] == "00")
    {
        cout << "\n";
        print_Tab(level);
        cout << "No files or folders are found!\n\n";
        return;
    }

    for (int i = 0; i < table_entry.size(); i++)
    {
        if (table_entry[i][0][0] != "00")
        {
            if (find_Attribute(table_entry[i]) == "Archive")
            {
                read_File(table_entry[i], level);
                cout << "\n\n";
            }
            else if (find_Attribute(table_entry[i]) == "Directory")
            {
                read_Folder(table_entry[i], level);
                cout << "\n\n";

                if (!is_Folder_Empty(table_entry[i]))
                {
                    read_RDet(find_table(device, get_First_Sector(table_entry[i]) * 512), level + 1);
                }
            }
        }
    }
```

Chia nội dung bảng thành các entry bằng hàm split_Entry. Nếu ở entry đầu bắt đầu bằng 00 thì không có entry (không có file, folder) nào được tạo.

Lặp qua các entry, nếu thuộc tính của entry là Archive (file) hoặc Directory(thư mục) thì đọc bằng hàm đọc tương ứng. Level dùng để phân cấp thư mục.

Nếu folder không rỗng thì thực hiện đệ quy in ra nội dung từng cấp folder.

4. NTFS

- Tương tự FAT32, thực hiện tạo một class NTFS để tổ chức và quản lý các thông tin và method cần thiết. Ngoài ra ta cần tạo thêm 1 class MFT_ENTRY để lưu thông tin của một entry trong bảng MFT. Class này chứa các thông tin như: recordID, parentID, type, name, data, size và một phương thức in nội dung entry ra màn hình.

a) MFT_ENTRY:

- Hàm printEntry: in các thông tin cơ bản của entry ra màn hình. Nếu loại entry là File thì lấy loại phần mềm tương thích từ util method getOpener. numTab phục vụ cho quá trình in cây thư mục

```
void MFT_ENTRY::printEntry(int numTab) {

    print_Tab(numTab);
    cout << "| " << "ID Record : " << this->recordID << " | " << endl;
    print_Tab(numTab);
    cout << "| " << "ID Parent: " << this->parentID << " | " << endl;
    print_Tab(numTab);
    cout << "| " << "Type of Entry: " << this->type << " | " << endl;
    print_Tab(numTab);
    cout << "| " << "Entry name: " << this->name << " | " << endl;
    if (this->type == "File")
    {
        string opener = getOpener(get_ext_file(this->name));
        if (opener.length() > 0)
        {
            print_Tab(numTab);
            cout << "| " << opener << " | " << endl;
        }
    }

    if (this->size >= 0)
    {
        print_Tab(numTab);
        cout << "| " << "Entry size: " << this->size << " bytes" << " | " << endl;
    }

    if (this->data.length() > 0)
    {
        print_Tab(numTab);
        cout << "| " << "Data: " << this->data << " | " << endl;
    }
    print_Tab(numTab);
    cout << "-------------------------------------" << endl;
}
```

b) NTFS

- Hàm read_BootSector: Đọc partition boot sector của ổ và in ra những thông tin cơ bản

```
void NTFS::readBoot_Sector()
{
    cout << "- Type of File System: NTFS" << endl;

    bytes_per_sector = hexstr_to_int(to_hexstr(BootSector, 11, 0, 2, 1));
    cout << "- The number of bytes per sector: " << bytes_per_sector << endl;

    sectors_per_cluster = hexstr_to_int(to_hexstr(BootSector, 13, 0, 1, 1));
    cout << "- The number of sectors per cluster: " << sectors_per_cluster << endl;

    cout << "- Media descriptor: " << to_hexstr(BootSector, 5, 1, 1, 1) << " hard disk" << endl;

    cout << "- Total number sectors NTFS: " << hexstr_to_int(to_hexstr(BootSector, 8, 2, 8, 1)) << endl;

    begin_MFT = hexstr_to_int(to_hexstr(BootSector, 0, 3, 8, 1));
    cout << "- The MFT begin at cluster: " << begin_MFT << endl;

    cout << "- MFT mirror begin at: " << hexstr_to_int(to_hexstr(BootSector, 8, 3, 8, 1)) << endl;
}
```

Loại hệ thống file, số byte trên 1 sector, số sector trên 1 cluster, media descriptor, số sector, cluster bắt đầu bảng MFT, cluster bắt đầu bảng MFT mirror

- Hàm get_type_file: Lấy loại entry (file/folder/unknown)

```
string NTFS::get_type_file(string flag_type)
{
    string type_file;
    if (flag_type == "10000000")
    {
        type_file = "Folder";
    }
    if (flag_type == "00000020")
    {
        type_file = "File";
    }
    return type_file;
}
```

- Hàm pushToMFTEntries: đẩy entry mới tìm được vào cuối của vector chứa các MFT entries

```
void NTFS::pushToMFTEntries(MFT_ENTRY entry)
{
    this->MFTEntries.push_back(entry);
}
```

- Hàm read_MFT: Đọc nội dung bảng MFT để tìm các entry

```
int id_record = hexstr_to_int(to_hexstr(sector, 12, 2, 4, 1));
newEntry.recordID = id_record;

int start_offset_$standard_info = hexstr_to_int(to_hexstr(sector, 4, 1, 2, 1));

int size_$standard_info = hexstr_to_int(to_hexstr(sector, (start_offset_$standard_info + 4) % 16, (start_offset_$standard_info + 4) / 16, 4, 1));

int start_offset_data_$standard_info = hexstr_to_int(to_hexstr(sector, (start_offset_$standard_info + 20) % 16, (start_offset_$standard_info + 20) / 16,
    + start_offset_$standard_info;

int start_offset_$file_name = start_offset_$standard_info + size_$standard_info;

int size_$file_name = hexstr_to_int(to_hexstr(sector, (start_offset_$file_name + 4) % 16, (start_offset_$file_name + 4) / 16, 4, 1));

int start_offset_data_$file_name = hexstr_to_int(to_hexstr(sector, (start_offset_$file_name + 20) % 16, (start_offset_$file_name + 20) / 16, 2, 1))
    + start_offset_$file_name;

int parent_id = hexstr_to_int(to_hexstr(sector, start_offset_data_$file_name % 16, start_offset_data_$file_name / 16, 6, 1));

newEntry.parentID = parent_id;

string flag_type = to_hexstr(sector, (start_offset_data_$file_name + 56) % 16, (start_offset_data_$file_name + 56) / 16, 4, 1);
string type = drive.get_type_file(flag_type);
newEntry.type = type;
```

Lấy các thông tin (recordID, parentID, type (loại file))

```
int len_file_name_attribute = hexstr_to_int(to_hexstr(sector, (start_offset_data_$file_name + 64) % 16, (start_offset_data_$file_name + 64) / 16, 1, 1));

int start_offset_file_name_attribute = start_offset_data_$file_name + 66;

int modun = start_offset_file_name_attribute % 16;

string file_name;

//doc ten file
if ((modun + len_file_name_attribute * 2) > 16) {
    file_name = hexstr_tostr(to_hexstr(sector, start_offset_file_name_attribute % 16, start_offset_file_name_attribute / 16,
        16 - modun, 1));
    reverse(file_name.begin(), file_name.end());
    int remain_size_file_name = len_file_name_attribute * 2 - (16 - modun);
    int now_offset = start_offset_file_name_attribute + (16 - modun);
    int q = remain_size_file_name / 16;
    int p = remain_size_file_name % 16;
    string name;
    for (int i = 0; i < q; i++)
    {
        name = hexstr_tostr(to_hexstr(sector, (now_offset + i * 16) % 16, (now_offset + i * 16) / 16,
            16, 1));
        reverse(name.begin(), name.end());
        file_name += name;
    }
    now_offset += q * 16;
    name = hexstr_tostr(to_hexstr(sector, now_offset % 16, now_offset / 16,
        p, 1));
    reverse(name.begin(), name.end());
    file_name += name;
}
else {
    file_name = hexstr_tostr(to_hexstr(sector, start_offset_file_name_attribute % 16, start_offset_file_name_attribute / 16,
        len_file_name_attribute * 2, 1));
    reverse(file_name.begin(), file_name.end());
}
newEntry.name = file_name;
```

Lấy thông tin tên entry

Nếu entry là file, lấy thông tin kích cỡ file, kiểm tra nếu là file txt thì lấy ra nội dung file

Sau khi lấy đầy đủ các thông tin cần thiết, kiểm tra loại file nếu đã được phân loại thì thêm vào vector MFT entries của NTFS. Clear bảng sector hiện tại để đọc tiếp thông tin.

- Hàm findSubDirectory: đệ quy qua vector MFTEntries để tìm entry có parentID tương ứng. In nội dung entry nếu tìm được parentID phù hợp. numTab để chia cây thư mục

```
void NTFS::findSubDirectory(int parentID, int numTab)
{

    for (int i = 0; i < MFTEntries.size(); i++)
    {
        MFT_ENTRY currentEntry = MFTEntries[i];
        if (currentEntry.parentID == parentID)
        {
            currentEntry.printEntry(numTab + 1);
            if (currentEntry.type == "Folder")
                findSubDirectory(currentEntry.recordID, numTab + 1);
        }
    }
}
```

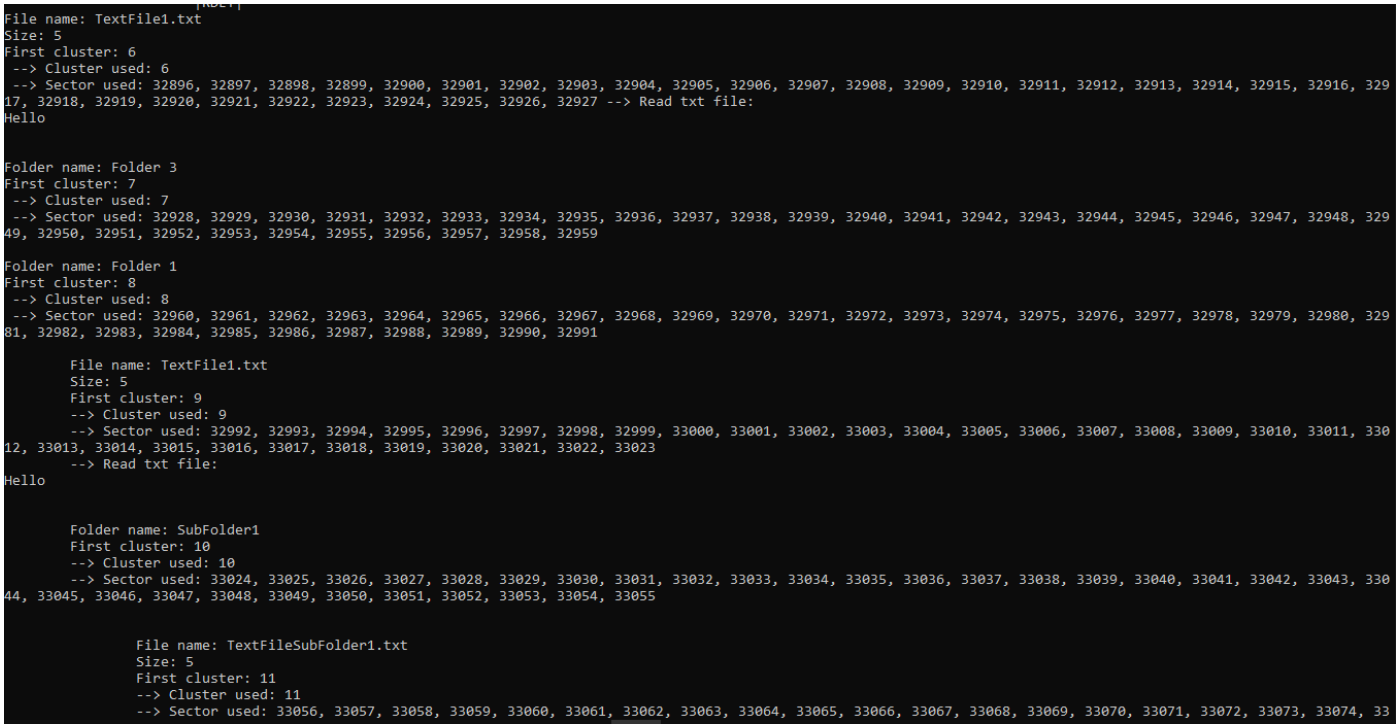
II. Demo tham khảo

1. **Video demo: https://www.youtube.com/watch?v=4Z5wua2W_jw**
2. **Hình ảnh demo:**

FAT32:

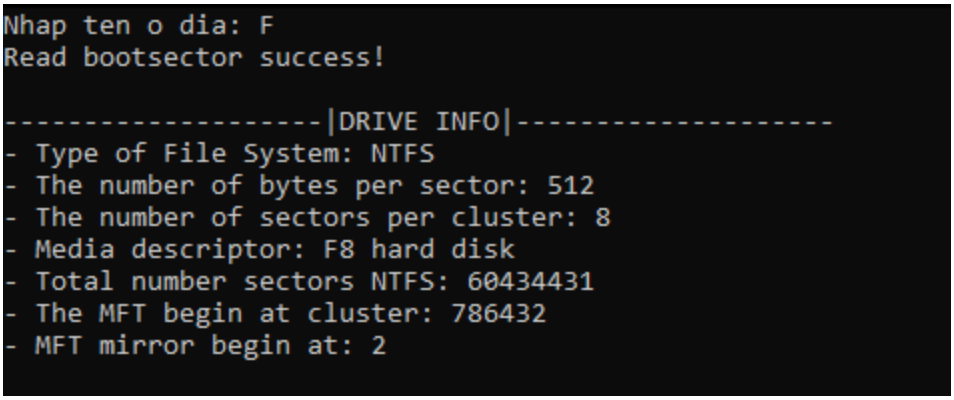
```
Nhap ten o dia: F
Read bootsector success!

--------------------|DRIVE INFO|--------------------
- Type of File System: FAT32
- The number of bytes per sector: 512
- The number of sectors per cluster: 32
- The number of bytes in Boot Sector: 3274
- The number of FAT table: 2
- Size of Volumn: 60434432
- Size of FAT table: 14747
- First sector of FAT table: 3274
- First sector of RDET: 32768
- First sector of Data: 32768
```

Thông tin ổ đĩa



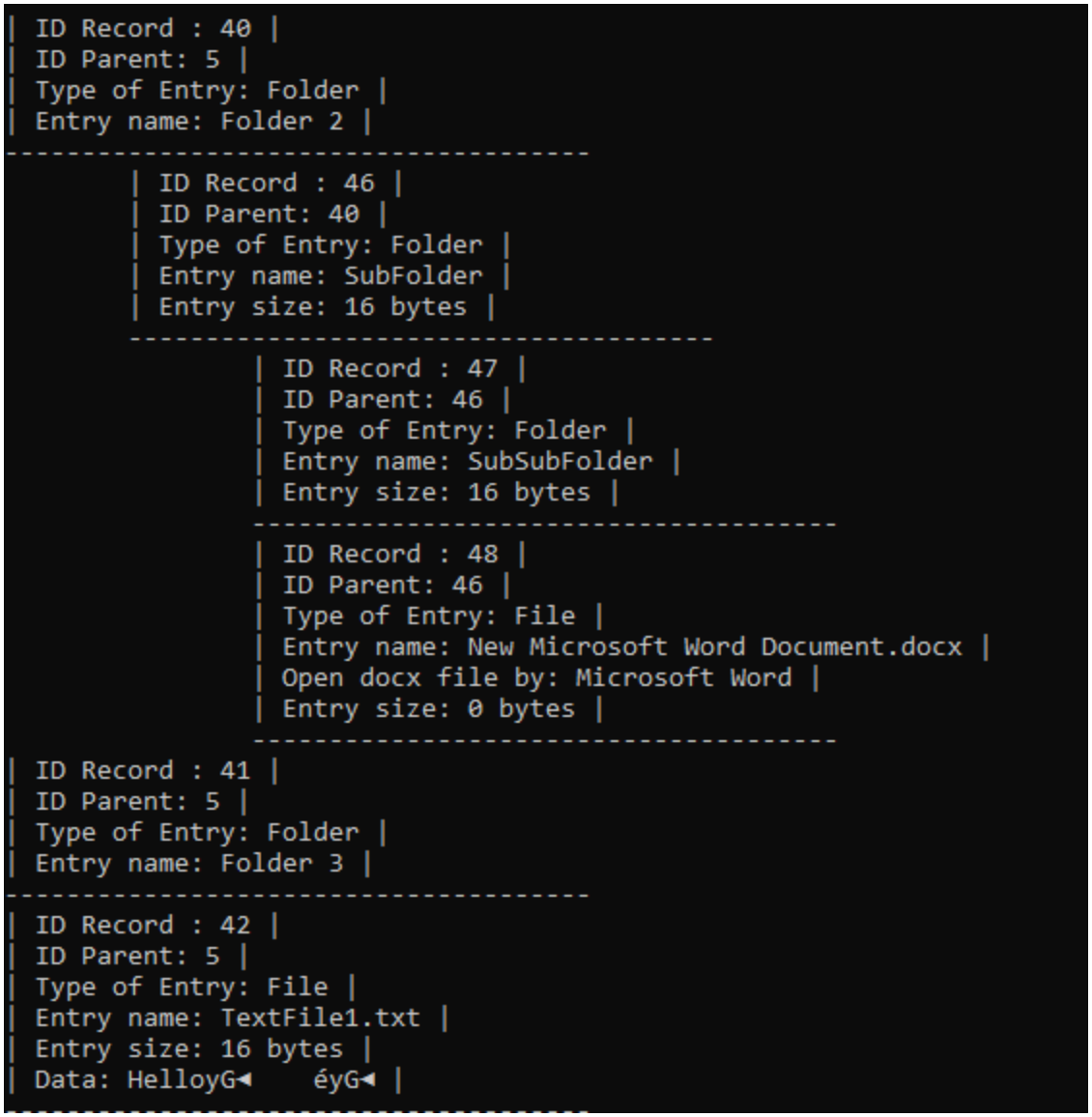
Cây thư mục

NTFS:



Thông số ổ đĩa

```
----------------------|MFT|----------------------
| ID Record : 39 |
| ID Parent: 5 |
| Type of Entry: Folder |
| Entry name: Folder 1 |
-------------------------------------
        | ID Record : 43 |
        | ID Parent: 39 |
        | Type of Entry: Folder |
        | Entry name: SubFolder1 |
        | Entry size: 16 bytes |
        ---------------------------------
                | ID Record : 44 |
                | ID Parent: 43 |
                | Type of Entry: File |
                | Entry name: TextFileSubFolder1.txt |
                | Entry size: 5 bytes |
                | Data: Hello |
                -------------------------------------
        | ID Record : 45 |
        | ID Parent: 39 |
        | Type of Entry: File |
        | Entry name: TextFile1.txt |
        | Entry size: 16 bytes |
        | Data: HelloyG◄    éyG◄ |
        ---------------------------------
| ID Record : 40 |
| ID Parent: 5 |
| Type of Entry: Folder |
| Entry name: Folder 2 |
-------------------------------------
        | ID Record : 46 |
        | ID Parent: 40 |
        | Type of Entry: Folder |
        | Entry name: SubFolder |
        | Entry size: 16 bytes |
        ---------------------------------
                | ID Record : 47 |
                | ID Parent: 46 |
                | Type of Entry: Folder |
                | Entry name: SubSubFolder |
                | Entry size: 16 bytes |
                -------------------------------------
```


| ID Record : 40 |
| ID Parent: 5 |
| Type of Entry: Folder |
| Entry name: Folder 2 |
| ID Record : 46 |
| ID Parent: 40 |
| Type of Entry: Folder |
| Entry name: SubFolder |
| Entry size: 16 bytes |
| ID Record : 47 |
| ID Parent: 46 |
| Type of Entry: Folder |
| Entry name: SubSubFolder |
| Entry size: 16 bytes |
| ID Record : 48 |
| ID Parent: 46 |
| Type of Entry: File |
| Entry name: New Microsoft Word Document.docx |
| Open docx file by: Microsoft Word |
| Entry size: 0 bytes |
| ID Record : 41 |
| ID Parent: 5 |
| Type of Entry: Folder |
| Entry name: Folder 3 |
| ID Record : 42 |
| ID Parent: 5 |
| Type of Entry: File |
| Entry name: TextFile1.txt |
| Entry size: 16 bytes |
| Data: HelloyG◄ éyG◄ |

Cây thư mục