**TRƯỜNG ĐẠI HỌC ĐÀ LẠT**

**KHOA CÔNG NGHỆ THÔNG TIN**

🙢❖🙠

**BÁO CÁO MÔN HỌC**

TÊN ĐỀ TÀI: “XUẤT NỘI DUNG CỦA THƯ MỤC BẤT KỲ TRÊN ĐĨA”

Giảng viên hướng dẫn: Đặng Thanh Hải

Sinh viên thực hiện: 2115208 – Lê Trung Hòa Hiếu

**Đà Lạt, tháng 6 năm 2023**

# Mục lục

[I. Mục lục 3](#_Toc138035065)

[II. Tổng quan 4](#_Toc138035066)

[1. Hệ điều hành 4](#_Toc138035067)

[2. Hệ thống quản lý tập tin 5](#_Toc138035068)

[3. Cấu trúc của ổ đĩa: 5](#_Toc138035069)

[4. Giới thiệu về chương trình 6](#_Toc138035070)

[III. Cài đặt chương trình 6](#_Toc138035071)

[1. Bảng tham số đĩa 6](#_Toc138035072)

[2. Cấu trúc FAT 7](#_Toc138035073)

[3. Entry của thư mục, tập tin 7](#_Toc138035074)

[4. Định nghĩa danh sách liên kết 9](#_Toc138035075)

[5. Biến toàn cục 10](#_Toc138035076)

[6. Hàm chức năng 10](#_Toc138035077)

[a. Chèn một nút vào danh sách 10](#_Toc138035078)

[b. So sánh hai chuỗi 10](#_Toc138035079)

[c. Phân tích đường dẫn và trích xuất thư mục: 11](#_Toc138035080)

[d. Tìm kiếm thư mục trong danh sách các entry của thư mục 13](#_Toc138035081)

[e. Đọc các sector của ổ đĩa lưu vào buffer 13](#_Toc138035082)

[f. Đọc bảng BPB 14](#_Toc138035083)

[g. Đổi các giá trị trong phân vùng thành side, track, sector tương ứng 14](#_Toc138035084)

[h. Đọc dữ liệu từ đĩa 15](#_Toc138035085)

[i. Đọc nội dung bảng FAT 15](#_Toc138035086)

[j. Tìm và trả về entry tiếp theo trong bảng FAT 15](#_Toc138035087)

[k. Lấy danh sách các cluster trong bảng FAT chứa phân vùng root 16](#_Toc138035088)

[l. Lấy danh sách các cluster liên tiếp trong bảng FAT 17](#_Toc138035089)

[m. Lấy danh sách các entry trong phân vùng 17](#_Toc138035090)

[n. In ra thông tin của các entry trong danh sách 18](#_Toc138035091)

[o. Gọi các hàm 19](#_Toc138035092)

[IV. Thực thi chương trình 20](#_Toc138035093)

[V. Đánh giá 21](#_Toc138035094)

[1. Thuận lợi 21](#_Toc138035095)

[2. Hạn chế 21](#_Toc138035096)

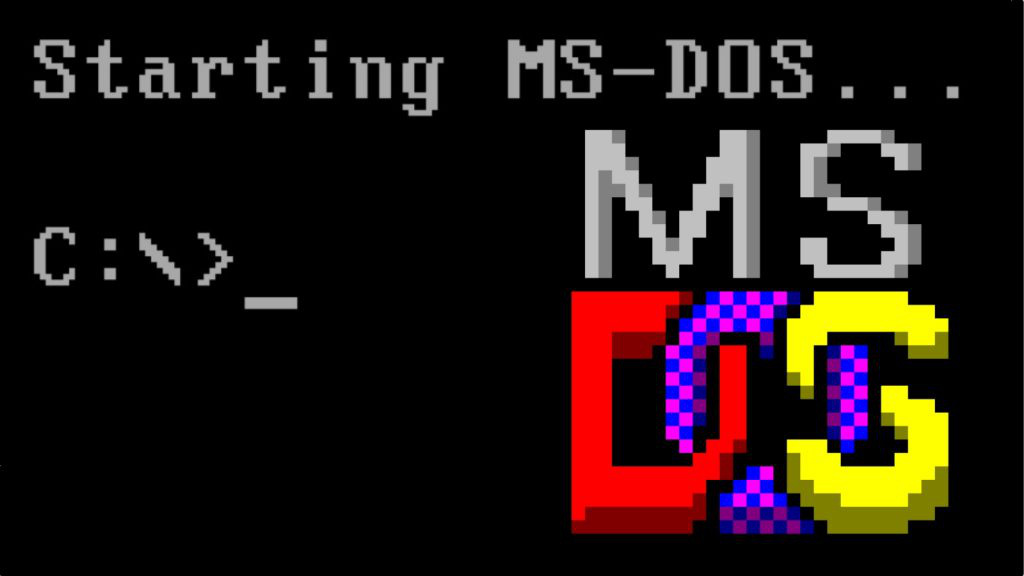
[VI. Tài liệu tham khảo 21](#_Toc138035097)

# Tổng quan

## Hệ điều hành

* Hệ điều hành là một chương trình được xem như trung gian giữa người sử dụng máy tính và phần cứng máy tính với mục đích thực hiện các chương trình giúp cho người dùng sử dụng máy tính dễ dàng hơn, sử dụng phần cứng một cách có hiệu quả.

Hình 1. Sơ đồ kiến trúc của Hệ điều hành

* Mục đích của hệ điều hành:
  + Thực hiện các chương trình giúp cho người dùng sử dụng máy tính dễ dàng hơn: Hệ điều hành cung cấp các ứng dụng và tiện ích để giúp người dùng thực hiện các tác vụ như xử lý văn bản, duyệt web, quản lý tập tin, và nhiều hơn nữa.
  + Giúp hệ thống máy tính thuận tiện trong việc sử dụng: Hệ điều hành cung cấp giao diện để người dùng có thể tương tác với máy tính một cách dễ dàng và thân thiện.
  + Sử dụng phần cứng một cách có hiệu quả: Hệ điều hành quản lý và điều khiển phần cứng của máy tính để đảm bảo rằng các phần mềm và ứng dụng hoạt động một cách chính xác và hiệu quả trên hệ thống. Nó cũng quản lý nguồn điện, bộ nhớ, bộ xử lý, và các thành phần khác của máy tính để đảm bảo rằng chúng hoạt động một cách hiệu quả và ổn định.
* MS-DOS (Microsoft Disk Operating System) là một hệ điều hành dòng lệnh độc quyền của Microsoft phát triển từ năm 1980 và được phát hành vào năm 1981. MS-DOS đã trở thành một trong những hệ điều hành phổ biến nhất trên các máy tính cá nhân vào những năm 1980 và 1990.
* MS-DOS cung cấp các lệnh cơ bản để quản lý tập tin, thư mục và đĩa, như lệnh DIR (hiển thị danh sách các file và thư mục), lệnh COPY (sao chép tập tin), lệnh DEL (xóa tập tin), và lệnh FORMAT (định dạng ổ đĩa). Bên cạnh đó, MS-DOS cũng cung cấp một số tiện ích và công cụ khác như DEBUG, EDIT và QBasic.

Hình 2. Giao diện và biểu tượng MS-DOS

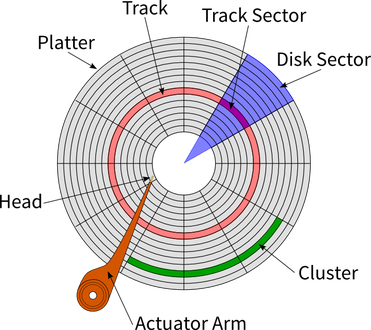
## Hệ thống quản lý tập tin

* Hệ điều hành MS-DOS sử dụng một hệ thống quản lý tập tin đơn giản để quản lý các file và thư mục trên ổ đĩa cứng. Hệ thống này được gọi là FAT (File Allocation Table).
* Trong hệ thống FAT, các file và thư mục được lưu trữ trên các cluster, là tập hợp các sector liên tiếp trên đĩa cứng. Bảng phân vùng (partition table) của ổ đĩa cứng chứa thông tin về cách phân vùng và tổ chức các cluster trên ổ đĩa cứng.

Hình 3. Cấu trúc FAT16

* Các lệnh cơ bản trong hệ thống quản lý tập tin của MS-DOS bao gồm:
  + DIR: Hiển thị danh sách các file và thư mục trên ổ đĩa cứng.
  + CD: Thay đổi thư mục hiện tại.
  + MD: Tạo thư mục mới.
  + RD: Xóa thư mục.
  + TYPE: Hiển thị nội dung của một file.
  + COPY: Sao chép file từ vị trí này sang vị trí khác.
  + DEL: Xóa file.
  + FORMAT: Định dạng ổ đĩa cứng.

## Cấu trúc của ổ đĩa:

* Cấu trúc của ổ đĩa bao gồm các thành phần sau:
  + Đĩa từ: Là một đĩa mỏng được làm từ nhựa, kim loại hoặc thủy tinh. Mỗi đĩa từ có thể sử dụng hai mặt (side).
  + Track: Là các vòng tròn đồng tâm trên mỗi mặt đĩa từ.
  + Sector: Là các đoạn hướng tâm của track, chứa thông tin được lưu trữ trên đĩa từ.
  + Cluster: Là tập hợp các sector liên tiếp trên đĩa từ. Các cluster được sử dụng để lưu trữ dữ liệu cho file hoặc thư mục.
  + Cylinder: Là tập hợp các track cùng bán kính trên các mặt đĩa khác nhau. Ví dụ, cylinder 0 chứa track 0 trên mặt đĩa từ thứ nhất và track 0 trên mặt đĩa từ thứ hai, cylinder 1 chứa track 1 trên hai mặt đĩa từ, và cứ như vậy.
* Các thành phần trên ổ đĩa cứng được quản lý và điều khiển bởi hệ điều hành, giúp cho các phần mềm và ứng dụng hoạt động một cách chính xác và hiệu quả trên hệ thống.

Hình 4. Cấu tạo của ổ đĩa

## Giới thiệu về chương trình

* Chương trình được xây dựng với mục đích cụ thể là giúp người dùng hiểu rõ hơn về cách thức hoạt động của hệ thống quản lý thông tin và cách cài đặt nó trên hệ điều hành DOS. Chương trình cho phép xuất nội dung bên trong của bất kỳ thư mục nào của đĩa, cùng với thông tin chi tiết của các tập tin và thư mục con bên trong.

# Cài đặt chương trình

Hệ thống tập tin sẽ được thiết kế dựa trên cấu trúc FAT16, với các thành phần chính như sau:

## Bảng tham số đĩa

Bảng tham số đĩa: một cấu trúc lưu trữ thông tin về đĩa và các thông tin cần thiết cho hệ thống tập tin, bao gồm số lượng sector, số lượng cluster, kích thước sector, kích thước cluster, số lượng sector dành riêng cho boot sector và số bảng FATs (thông số này xác định số lượng bảng FAT).

|  |  |  |  |
| --- | --- | --- | --- |
| **Offset (bytes)** | **Tên** | **Kích thước (bytes)** | **Ý nghĩa** |
| 0 | JMP | 3 | Lệnh nhảy đến đọan mồi HĐH (đoạn mồi: phần khởi động) |
| 3 | Ver | 8 | Nhận diện OEM (tên nhà sản xuất hệ thống tập tin) |
| 11 | SecSiz | 2 | Kích thước sector trong byte |
| 13 | ClustSiz | 1 | Số lượng sector mỗi cluster |
| 14 | ResSec | 2 | Số sector dành riêng cho boot sector |
| 16 | FatCnt | 1 | Số bảng FAT trên đĩa |
| 17 | RootSiz | 2 | Số entry trong thư mục gốc |
| 19 | TotSec | 2 | Tổng số sector trên đĩa nhỏ hơn 32MB |
| 21 | Media | 1 | Loại đĩa |
| 22 | FatSiz | 2 | Số sector cho mỗi bảng FAT |
| 24 | TrkSec | 2 | Số sector trên mỗi track |
| 26 | HeadCnt | 2 | Số head trên đĩa |
| 28 | HidSec | 4 | Số sector ẩn |
| 32 | TotSec | 4 | Tổng số sector lớn hơn 32 MB |

## Cấu trúc FAT

Cấu trúc FAT (File Allocation Table): một bảng lưu trữ các thông tin địa chỉ các ô nhớ còn trống, các ô nhớ nào thuộc về mỗi tập tin, số lượng sector của mỗi file và số lượng cluster của mỗi file.

|  |  |
| --- | --- |
| **Giá trị entry** | **Mô tả** |
| 0(000) | Cluster còn trống |
| (0)002 – (F)FEF | Chứa dữ liệu, giá trị của nó là vị trí của cluster tiếp theo |
| (F)FF0 – (F)FF6 | Dành riêng, không sử dụng |
| (F)FF7 | Bad sector |
| (F)FF8 – (F)FFF | Cluster cuối của một tập tin |

## Entry của thư mục, tập tin

Cấu trúc của mỗi Entry tập tin: mỗi Entry tập tin sẽ bao gồm các thông tin như tên file, địa chỉ cluster đầu tiên của file, kích thước file và thuộc tính của file (ví dụ: read-only, hidden, system...).

|  |  |  |
| --- | --- | --- |
| **Offset** | **Kích thước** | **Ý nghĩa** |
| +0h | 8 | Tên tập tin hay thư mục |
| +8h | 3 | Phần mở rộng tên tập tin |
| +0bh | 1 | Thuộc tính tập tin |
| +0ch | 0ah | Dành riêng |
| +16h | 1 | Thời gian tạo |
| +18h | 2 | Ngày tạo |
| +1ah | 2 | Số hiệu cluster đầu tiên |
| +1ch | 4 | Kích thước tập tin (bytes) |

**Cài đặt:**

// BIOS Parameter Block (BPB)

typedef struct

{

    char jmp[3];      // The first byte to JMP SHORT XX NOP

    char Ver[8];      // OEM identifier

    unsigned SecSiz;  // Bytes/sector (in little-endian format)

    char ClustSiz;    // Sectors/cluster

    unsigned ResSec;  // Number of reserved sectors (included Boot Record Sectors)

    char FatCnt;      // Number of File Allocation Tables. Often 2

    unsigned RootSiz; // Max # of dictionary entries

    unsigned TotSec;  // Total logical sectors

    char Media;       // Indicates the media descriptor type (FAT ID)

    unsigned FatSiz;  // Number of sectors per FAT

    unsigned TrkSec;  // Number of sectors per track

    unsigned HeadCnt; // Number of heads or sides on the storage media

    unsigned HidSec;  // Number of hidden sectors (LBA)

} EntryBPB;

// Union between Sector array and BPB structure

typedef union

{

    char Sec[512];

    EntryBPB Entry;

} UnionBPB;

// The file time

typedef struct

{

    unsigned S : 5; // Seconds

    unsigned M : 6; // Minutes

    unsigned H : 5; // Hours

} Time;

// Union between Creation time and Time structure

typedef union

{

    unsigned intTime;

    Time T;

} UnionTime;

// The file date

typedef struct

{

    unsigned D : 5; // Days

    unsigned M : 4; // Months

    unsigned Y : 7; // Years

} Date;

// Union between Creation date and Date structure

typedef union

{

    unsigned intDate;

    Date Dat;

} UnionDate;

// Attribute Byte (8 bits)

typedef struct

{

    char ReadOnly : 1; // Indicates that the file is read only

    char Hidden : 1;   // Indicates a hidden file

    char System : 1;   // Indicates a system file

    char Volume : 1;   // Indicates a special entry containing the disk's volume label

    char SubDir : 1;   // Describes a subdirectory

    char Archive : 1;  // This is the archive flag

    char DR : 2;

} Attribute;

// Union between Attribute char and Attribute structure

typedef union

{

    char charAtt;

    Attribute Attr;

} UnionAttribute;

// Regular Directory Entry

typedef struct

{

    char FileName[8];   // The file name, the first byte indicates it's status

    char Ext[3];        // The file name extension

    UnionAttribute Att; // The file attributes

    char DR[10];

    UnionTime Tg;   // The file time

    UnionDate Ng;   // The file date

    unsigned Clust; // The starting cluster number

    long FileSize;  // The file size

} EntryDir;

// Union between Entry char array and Entry structure

typedef union

{

    char Entry[32];

    EntryDir Entdir;

} UnionDir;

## Định nghĩa danh sách liên kết

// Define a node in linked list

typedef struct Node

{

    void \*Data; // Content of node

    Node \*Next; // The pointer of next node

} NodeType;

// Define a linked list

typedef NodeType \*PointerType;

## Biến toàn cục

char \*path;         // Duong dan

char drive;         // O dia

EntryBPB BPB;       // Bang tham so dia

unsigned char \*FAT; // Noi dung bang FAT

long SecStart = 0;  // Sector bat dau

## Hàm chức năng

### Chèn một nút vào danh sách

/\*Chen mot nut vao danh sach

Tham so dau vao:

- list: Danh sach lien ket

- lastItem: Phan tu cuoi cung cua danh sach

- item: Phan tu can chen

Tra ve:

- true: Thanh cong

- false: That bai

\*/

int InsertLast(PointerType &list, PointerType &lastItem, void \*item)

{

    PointerType temp = new NodeType;

    if (!temp) // Kiem tra viec cap phat bo nho

    {

        cout << "\nLoi cap phat bo nho!";

        return 0;

    }

    temp->Data = item; // Gan gia tri cua item cho con tro Data cua temp

    temp->Next = NULL;

    if (list == NULL) // Kiem tra danh sach lien ket

        list = temp;

    else

        lastItem->Next = temp;

    lastItem = temp;

    return 1;

}

### So sánh hai chuỗi

/\*So sanh hai chuoi

Tham so dau vao:

- string1: Chuoi ban dau

- string2: Chuoi thu hai

Tra ve:

- true: bang nhau

- false: khong bang nhau

\*/

int Compare(char \*string1, char \*string2)

{

    for (int i = 0; string1[i] != '\0'; i++) // Duyet qua tung ky tu cua string1

        if (string1[i] != string2[i]) // Neu co ky tu trong string1 khac string2

            return 0;

    if (string2[i] == ' ') // Kiem tra ky tu tiep theo co phai la khoang trang hay khong

        return 1;

    else

        return 0;

}

### Phân tích đường dẫn và trích xuất thư mục:

/\*Phan tich duong dan va trich xuat ten tep/thu muc

Tham so dau vao:

- path: Chuoi duong dan ma nguoi dung nhap vao

Tra ve:

- NULL: Duong dan khong hop le

- !NULL: Danh sach duong dan

\*/

PointerType AnalysePath(char \*path)

{

    PointerType list = NULL, last = NULL; // Khai bao con tro den nut dau tien va cuoi cung cua danh sach lien ket

    char \*fileName; // Luu ten tep/thu muc trong duong dan

    drive = path[0]; // Trich xuat chu cai o dia bang ky tu dau tien

    if (drive >= 97 && drive <= 122) //Neu la chu thuong tru di 97 de thanh chu hoa, chuyen gia tri chu thanh gia tri so tuong ung

        drive -= 97;

    else if (drive >= 65 && drive <= 90) // Neu la chu hoa tru 65, tuong tu

        drive -= 65;

    else // Neu ky tu dau khong phai la chu cai

    {

        cout << "\nKhong co o dia " + drive;

        return NULL;

    }

    if (drive == 2)

        drive = 0x80; //Gia tri o C

    if (drive == 0 || drive == 1 || drive == (char)0x80) // Chi chap nhan o A, B, C

    {

        if (path[1] != ':' && path[2] != '\\') // Kiem tra ky tu tiep theo co phai la :\  //

        {

            cout << "\nDuong dan nhap sai";

            return NULL;

        }

        if (path[3] == '\0') // Kiem tra null

        {

            cout << "\nDuong dan chua co thu muc";

            return NULL;

        }

        int i = 3; // 4 ky tu con lai

        while (path[i] != '\0') // Doc den het duong dan

        {

            int j = 0;

            fileName = new char[12];

            if (!fileName)

            {

                cout << "\nLoi cap phat bo nho cho ten file";

                return NULL;

            }

            while (path[i] != '\\' && path[i] != '\0') // Doc den het ten cua thu muc

            {

                fileName[j] = path[i];

                ++i;

                ++j;

            }

            fileName = strupr(fileName); // Doi sang chu hoa

            fileName[j] = '\0';

            j = 0;

            while (fileName[j] != '.' && fileName[j] != '\0') // Lay ten cua thu muc (khong tinh phan mo rong)

                ++j;

            if (fileName[j] == '.') // Neu no la tap tin, chuyen doi thanh quy uoc dat ten 8.3 (8: ten, 3: phan mo rong)

            {

                fileName[j] = ' ';

                ++j;

                int k = 3;

                while (fileName[j] != '\0')

                {

                    fileName[11 - k] = fileName[j];

                    ++j;

                }

                fileName[11] = '\0';

                k = 7;

                while (fileName[k] != ' ')

                {

                    fileName[k] = ' ';

                    k--;

                }

            }

            if (path[i] != '\0')

                i++;

            InsertLast(list, last, fileName);

        }

        return list;

    }

    else

    {

        cout << "\nDuong dan thieu o dia!";

        return NULL;

    }

}

### Tìm kiếm thư mục trong danh sách các entry của thư mục

/\*Tim kiem thu muc trong danh sach cac entry cua thu muc

Tham so dau vao:

- listEntry: Danh sach entry cua mot thu muc (goc hoac con)

- fileName: Ten cua thu muc con

- dir: Thu muc tim duoc (ket qua)

Tra ve:

- true: Tim thay

- false: Khong tim thay

\*/

int SearchDir(PointerType listEntry, char \*fileName, EntryDir &dir)

{

    PointerType p = listEntry;

    while (p != NULL) // Duyet toan bo entry cua thu muc

    {

        if (Compare(fileName, ((EntryDir \*)p->Data)->FileName))

        {

            dir = \*(EntryDir \*)p->Data;

            return 1;

        }

        p = p->Next;

    }

    return 0;

}

### Đọc các sector của ổ đĩa lưu vào buffer

/\*Doc cac sector cua o dia luu vao buffer bang ham ngat 13 cua BIOS

Tham so dau vao

- buff: Con tro toi buffer de luu du lieu doc vao

- side: So mat cua dia

- track: So track cua dia

- sector: Vi tri sector

- number: So luong sector can doc

Tra ve:

- true: Thanh cong

- false: Khong thanh cong

\*/

int ReadDiskBIOS(char \*buff, unsigned side, unsigned track, unsigned sector, unsigned number)

{

    union REGS u, v; // Thanh ghi da dung

    struct SREGS s;  // Thanh ghi doan

    int i = 0;    // i: So lan doc toi da

    v.x.cflag = 1;   // Dat co carry trong thanh ghi v thanh 1, de kiem tra loi khi doc dia

    while (i < 2 && v.x.cflag != 0)

    {

        //Chi dinh cac tham so can thiet de doc tu dia

        u.h.ah = 0x2;   // Doc dia - Neu la 0x1: Ghi dia

        u.h.dl = drive; // O dia - Neu la 80: Dia cung

        u.h.dh = side;            // Side

        u.h.cl = sector;          // Sector bat dau doc du lieu

        u.h.ch = track;           // Track

        u.x.cx = track;           // Track bat dau doc du lieu

        u.h.al = number;          // So luong sector can doc

        s.es = FP\_SEG(buff);      // Vi tri buff luu du lieu duoc doc

        u.x.bx = FP\_OFF(buff);    // Dia chi offset cua buff

        int86x(0x13, &u, &v, &s); // Ham ngat 13h

        i++;                      // Tang so lan doc

    }

    return !(v.x.cflag); // Kiem tra loi

}

### Đọc bảng BPB

/\*Doc bang tham so dia BPB va luu vao bien toan cuc BPB

 \*/

void ReadBPB()

{

    UnionBPB temp;

    if (drive == 0 || drive == 1 || drive == (char)0x80) //Xac dinh o dia doc la dia mem hay dia cung

        if (!ReadDiskBIOS(temp.Sec, 0, 1, 1, 1))

        {

            cout << "\nKhong doc duoc bang tham so dia";

            return;

        }

    BPB = temp.Entry;

}

### Đổi các giá trị trong phân vùng thành side, track, sector tương ứng

/\*Doi cac gia tri trong phan vung thanh cac thong so side, track, sector tuong ung

Tham so dau vao:

- begin: Vi tri bat dau

- side: Mat cua dia

- track: So track cua dia

- sector: Vi tri sector

Gia tri track moi la ket hop giua gia tri sector

Luu y: Phai gan cho bien toan cuc BPB truoc

\*/

void Change(long begin, unsigned &side, unsigned &track, unsigned &sector)

{

    unsigned x;

    sector = (unsigned)(1 + begin % BPB.TrkSec);

    side = (unsigned)((begin / BPB.TrkSec) % BPB.HeadCnt);

    track = (unsigned)(begin / (BPB.TrkSec \* BPB.HeadCnt));

    x = track; // Gia tri track ban dau

    x = x & 0xFF00; // Lam tron den 8 bit cao nhat

    x = x >> 2; // Dich sang phai 2 bit

    x = x & 0x00FF; // Lam tron 8 bit thap nhat

    x = x | sector;  // Ket hop voi gia tri cua sector de tao ra gia tri moi cho track

    track = track << 8; // Dich sang trai 8 bit

    track = track | x;

}

### Đọc dữ liệu từ đĩa

/\*Doc du lieu tu dia

Tham so dau vao:

- buff: Buff luu du lieu doc duoc

- begin: Vi tri bat dau doc

- number: So sector can doc

Tra ve:

- true: Thanh cong

- false: That bai

\*/

int ReadDisk(char \*buff, long begin, int number)

{

    unsigned side, track, sector;

    Change(begin, side, track, sector);

    if (ReadDiskBIOS(buff, side, track, sector, number))

        return 1;

    else

        return 0;

}

### Đọc nội dung bảng FAT

/\*Doc noi dung bang FAT cho vao bien toan cuc

Luu y: Phai gan gia tri cho bien toan cuc BPB truoc

\*/

int ReadFAT()

{

    FAT = new char[BPB.FatSiz \* BPB.SecSiz];

    if (!FAT)

    {

        cout << "\nLoi cap phat bo nho!";

        return 0;

    }

    if (!ReadDisk(FAT, BPB.ResSec, BPB.FatSiz)) // Doc noi dung trong bang FAT

    {

        cout << "\nKhong doc duoc bang tham so dia!";

        return 0;

    }

    return 1;

}

### Tìm và trả về entry tiếp theo trong bảng FAT

/\*Tim va tra ve entry tiep theo trong bang FAT

Tham so dau vao:

- index: Entry hien tai trong bang FAT

Tra ve: Vi tri cluster tiep theo cua index

Luu y: Phai gan ket qua cho bien toan cuc FAT truoc

\*/

unsigned NextEntry(unsigned index)

{

    unsigned add, x, t;

    if (drive == 0 || drive == 1) // Neu o dia hien tai la dia mem (A, B)

    {

        add = index \* 3 / 2; // Tinh toan vi tri byte dau tien cua 2 entry lien tiep

        x = FAT[add]; // Luu gia tri cua entry trong byte thu nhat

        t = FAT[add + 1]; // Luu gia tri cua entry trong byte lien tiep

        t = t << 8; // Dich sang trai 8 bit

        x = x + t; // Tao ra gia tri entry tiep theo

        if ((index % 2) == 0)

            x = x & 0x0FFF; // Neu la chan, lam tron xuong 12 bit thap nhat

        else

            x = x >> 4; // Neu la le, dich sang phai 4 bit

    }

    else // Neu khong phai la dia mem

    {

        add = (index \* 2) - 1; // Vi tri byte dau tien cua entry tiep theo

        //Xu ly tuong tu dia mem

        x = FAT[add];

        t = FAT[add + 1];

        t = t << 8;

        x = x + t;

    }

    return x;

}

### Lấy danh sách các cluster trong bảng FAT chứa phân vùng root

/\*Lay danh sach cac cluster trong bang FAT chua phan vung root

Tra ve: Danh sach cluster

Luu y: Phai gan ket qua cho bien toan cuc BPB truoc

\*/

PointerType GetClusterRoot()

{

    PointerType listCluster = NULL, last = NULL;

    unsigned rootSec; //Luu vi tri sector dau tien trong phan vung root

    unsigned \*cluster;

    unsigned number; // So luong sector su dung de luu tru cac entry trong phan vung root

    rootSec = BPB.ResSec + BPB.FatSiz \* (int)BPB.FatCnt;

    number = BPB.RootSiz \* 32 / 512;

    for (int i = 0; i < number; i++, rootSec++)

    {

        cluster = new unsigned;

        \*cluster = rootSec;

        InsertLast(listCluster, last, (unsigned \*)cluster);

    }

    return listCluster;

}

### Lấy danh sách các cluster liên tiếp trong bảng FAT

/\*Lay danh sach cac cluster lien tiep trong bang FAT

Tham so dau vao:

- start: Vi tri cluster bat dau

Tra ve: Danh sach cluster

\*/

PointerType GetCluster(unsigned start)

{

    PointerType listCluster = NULL, last = NULL;

    unsigned \*cluster, next = start;

    while(next >= 2 && next < 0xFEF ) // 2 la entry dau tien cua cluster, 0xFEF la entry cuoi cung

    {

        cluster = new unsigned;

        \*cluster = next;

        InsertLast(listCluster, last, (unsigned \*)cluster);

        next = NextEntry(\*cluster);

    }

    return listCluster;

}

### Lấy danh sách các entry trong phân vùng

/\*Lay danh sach cac entry trong phan vung

Tham so dau vao:

- listCluster: Danh sach cluster

- flag:     0 : Thu muc root

            1 : Thu muc con

Tra ve:

- NULL: Khong thanh cong

- !NULL: Danh sach entry cua thu muc

Luu y: Phai gan gia tri cho bien toan cuc BPB, drive truoc

\*/

PointerType GetEntryDir(PointerType listCluster, char flag)

{

    PointerType p = listCluster, listEntry = NULL, last = NULL;

    unsigned size = BPB.ClustSiz \* 512; // Kich thuoc cluster

    unsigned currentCluster, currentSector; // Cluster hien tai, Sector dau tien cua cluster hien tai

    unsigned char \*buffDir = new unsigned char[size]; // Khoi tao buffer co kich thuoc la size

    UnionDir \*dir; // Du lieu cac entry doc duoc

    while (p != NULL)

    {

        currentCluster = \*(unsigned \*)p->Data;

        if (flag)

            currentSector = BPB.ResSec + BPB.FatSiz \* BPB.FatCnt + (BPB.RootSiz \* 32) / 512 + (currentCluster - 2) \* BPB.ClustSiz;

        else

            currentSector = currentCluster;

        if (!ReadDisk(buffDir, currentSector, BPB.ClustSiz))

        {

            cout << "\nKhong doc duoc dia!";

            return NULL;

        }

        for (int i = 0; i < size;)

        {

            if (buffDir[i] != (char)0x00)     // Kiem tra co phai entry rong

                if (buffDir[i] != (char)0xE5) // Neu entry khong bi xoa

                {

                    int j = 0;

                    dir = new UnionDir;

                    for (; j < 32; j++, i++)

                        dir->Entry[j] = buffDir[i];

                    InsertLast(listEntry, last, &dir->Entdir);

                }

                else

                    i += 32;

            else

                break;

        }

        p = p->Next;

    }

    delete buffDir;

    return listEntry;

}

### In ra thông tin của các entry trong danh sách

/\*In ra thong tin cua cac entry trong danh sach

Tham so dau vao:

- listEntry: Danh sach entry can hien thi thong tin

\*/

void PrintTo(PointerType listEntry)

{

    PointerType p = listEntry;

    EntryDir dir;

    cout << '\n'

         << setiosflags(ios::left)

         << setw(8) << "Name"

         << setw(8) << "Ext"

         << setw(16) << "Size (bytes)"

         << setw(8) << "Time"

         << setw(12) << "Date";

    while (p != NULL)

    {

        dir = \*(EntryDir \*)p->Data;

        if (dir.Att.Attr.Volume == 0)

        {

            int i;

            cout << "\n";

            for (i = 0; i < 8; i++)

                cout << dir.FileName[i];

            for (i = 0; i < 3; i++)

                cout << dir.Ext[i];

            cout << '\t' << dir.FileSize

                 << "\t\t" << ((dir.Tg).T).H << ':' << ((dir.Tg).T).M

                 << '\t' << ((dir.Ng).Dat).D << '/' << ((dir.Ng).Dat).M << "/" << ((dir.Ng).Dat).Y + 1980;

        }

        p = p->Next;

    }

}

### Gọi các hàm

void ChayChuongTrinh()

{

    path = new char[256];

    cout << "\nNhap duong dan: ";

    scanf("%s", path);

    PointerType listPath = AnalysePath(path);

    if (listPath == NULL)

        return;

    ReadBPB(); // Doc bang BPB cua phan vung dang xet

    if (!ReadFAT()) // Doc bang FAT cua phan vung dang xet

        return;

    // Tai thu muc goc

    PointerType listCluster = GetClusterRoot();

    PointerType listEntry = GetEntryDir(listCluster, 0);

    // Tai thu muc con

    while (listPath)

    {

        EntryDir dir;

        if (SearchDir(listEntry, (char \*)listPath->Data, dir))

        {

            listCluster = GetCluster(dir.Clust);

            listEntry = GetEntryDir(listCluster, 1);

        }

        else

        {

            cout << "\nKhong tim thay thu muc " << (char \*)listPath->Data;

            return;

        }

        listPath = listPath->Next;

    }

    PrintTo(listEntry);

}

int main()

{

    char key;

    clrscr();

    do

    {

        ChayChuongTrinh();

        key = getch();

        cout<<"Nhan ESC de thoat khoi chuong trinh.";

    } while (key != 27); //27 la ma ASCII cua phim ESC

    return 1;

}

# Thực thi chương trình

* Kết quả:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

* So sánh với lệnh DIR của MS-DOS:

A screenshot of a computer

Description automatically generated

# Đánh giá

## Thuận lợi

- Việc cài đặt chương trình và hiệu chỉnh các hàm để phù hợp với mục đích sử dụng không phải là quá khó khăn. Hầu hết các hàm được sử dụng trong chương trình đã được nghiên cứu và xây dựng từ trước, do đó việc thực hiện các hàm này không gây khó khăn cho người dùng.

- Tuy nhiên, người viết chương trình cũng đã phải tiến hành chỉnh sửa để tối ưu hóa chương trình và khắc phục các cảnh báo khi biên dịch chương trình. Điều này giúp tăng hiệu suất của chương trình và đảm bảo rằng không có lỗi nào xảy ra trong quá trình sử dụng.

## Hạn chế

Hiện tại, chương trình chỉ mới xuất được kích thước của tập tin, chưa xuất được kích thước của thư mục.

# Tài liệu tham khảo

[1] Giáo trình Hệ điều hành, Đặng Thanh Hải, Khoa Công nghệ thông tin, Trường Đại học Đà Lạt

[2] MS-DOS, Encyclopædia Britannica, <https://www.britannica.com/topic/MS-DOS>

[3] FAT File Sytem, <https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-2000-server/cc938438(v=technet.10)?redirectedfrom=MSDN>