МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Кафедра прикладної математики

ДОСЛІДЖЕННЯ ЛОГІЧНОЇ СТРУКТУРИ ДИСКУ В MS-DOS

методичні рекомендації до виконання лабораторної роботи №1

із дисципліни:

«Операційні системи»

Київ — 2015

Зміст

ВСТУП

Операційна система (ОС) MS-DOS (Microsoft Disk Operating System) надає користувачеві низку можливостей для роботи з файлами даних, їх організації в каталоги та використання пристроїв уведення-виведення. MS-DOS є однозадачною однокористувацькою операційною системою, що працює в реальному режимі мікропроцесорів x86. Вона використовує 640 КБ пам’яті комп’ютера та підтримує відносно просту файлову систему.

Історично MS-DOS було орієнтовано на роботу з мікропроцесорами з одним режимом роботи — реальним (мікропроцесори 8086 та 8088). Захищений режим роботи (мікропроцесори 80286 та вище) з адресацією до 16 МБ пам’яті можуть використовувати тільки деякі драйвери MS-DOS. Операційна система не працює з віртуальною пам’яттю.

Найдосконалішою версією операційної системи MS-DOS є версія MS-DOS 6.22, випущена окремо в 1994 р. Роботу з цією системою розглядатимемо в даній лабораторній роботі.

1 ПОСТАНОВКА ЗАДАЧІ

У даній лабораторній роботі потрібно виконати аналіз логічної структури диску в операційній системі MS-DOS 6.22, у тому числі вивчити системну область диску, навчитися працювати з таблицею розподілення файлів та вивчати вміст кореневого каталогу.

У рамках виконання лабораторної роботи потрібно:

а) ознайомитися з теоретичними відомостями, викладеними в розділі 2;

б) написати будь-якою мовою програмування програму, яка повинна виконувати такі дії:

1) зчитувати завантажувальний сектор гнучного диску в ОС MS-DOS 6.22 та виводити на екран його вміст у зручному форматі;

2) зчитувати кореневий каталог гнучного диску в ОС MS-DOS 6.22 та виводити на екран список дескрипторів файлів та підкаталогів кореневого каталогу; у випадку великої кількості файлів/підкаталогів повинно бути передбачено можливість посторінкового виведення;

3) виводити на екран список усіх кластерів, розподілених деякому файлу/підкаталогу кореневого каталогу; у програмі повинно бути передбачено можливість вибору з боку користувача такого файлу/підкаталогу; у випадку великої кількості кластерів повинно бути передбачено можливість посторінкового виведення;

в) відлагодити програму в ОС MS-DOS 6.22;

г) підготувати звіт із лабораторної роботи відповідно до вимог розділу 3.

2 ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1 Файлова структура диску в MS-DOS

За наявності великої кількості програм та даних потрібні їх чіткий облік та систематизація. Операційним системам доводиться працювати з різними потоками даних, апаратними й периферійними пристроями комп'ютера. Організувати впорядковане керування всіма ціма об'єктами дозволяє *файлова система* — система керування даними, яка забезпечує роботу аплікацій із зовнішніми пристроями пам'яті.

Для забезпечення доступу до файлів файлова система MS-DOS організує й підтримує на логічному диску певну *файлову структуру*.

Файлова система використовує декілька рівнів представлень про структуру дискового простору, кожен із яких орієнтовано на певну групу користувачів. Кожен рівень представлень дозволяє розв'язувати певне коло задач доступу до файлів.

За цим критерієм розрізняють *інформаційну*, *фізичну* та *логічну* структури, для котрих існує певний набір структурних елементів та відповідна термінологія.

2.1.1 Інформаційна структура дискового простору

Інформаційна структура дискового простору — це зовнішнє представлення дискового простору, орієнтоване на користувача та визначуване такими елементами, як *том* (логічний диск), *каталог* (папка, директорія)

та *файл*. Ці елементи використовують під час спілкування користувача з операційною системою. Спілкування здійснюють за допомогою команд, які виконують операції доступу до файлів та каталогів.

Дисковий простір представлено користувачу як множину *логічних дисків*, для позначення яких використовують літери латинського алфавіту. При цьому імена A та B зарезервовано для гнучних дисків.

Із кожним логічним диском пов'язано *дерево каталогів*. Дерево каталогів *обов'язково* містить один *кореневий каталог* (root directory) та множину єрархічно підпорядкованих каталогів. На відформатованому диску завжди існує кореневий каталог. Розмір такого каталогу для даного диску — величина фіксована, тому максимальну кількість «прив'язаних» до нього файлів та дочірніх каталогів строго визначено. Кореневий каталог не має імені, можна вважати, що його ім'я збігається з іменем відповідного логічного диску.

Підпорядковані каталоги — це файли певної структури, аналогічної структурі кореневого каталогу. Розмір підпорядкованого каталогу не фіксований: він змінюється динамічно з додаванням та видаленням реєстрованих у ньому об'єктів (файлів та дочірніх підкаталогів). Розмір підпорядкованого каталогу обмежено тільки розміром логічного диску.

MS-DOS підтримує єрархічну (деревоподібну) структуру каталогів, представлену на рисунку 2.1. На відміну від кореневого каталогу, інші підкаталоги створюють за допомогою спеціальних внутрішніх команд MS-DOS. Основна ціль такої структури каталогів — організація ефективного збереження великої кількості файлів на диску.

Кожний каталог, окрім кореневого, має батька, тобто кожний каталог, окрім кореневого, має інший каталог, до якого його «прив'язано» (інколи кажуть «зареєстровано»). На рисунку 2.1 «Каталог 2.1» прив'язано до

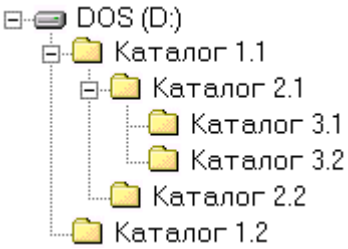

Рисунок 2.1 – Єрархічна структура каталогів в MS-DOS

«Каталог 1.1», тобто він є дочірнім відносно «Каталог 1.1» та батьківським — відносно «Каталог 3.1» та «Каталог 3.2». MS-DOS розглядає кожний каталог, окрім кореневого, як файл.

2.1.2 Фізична структура дискового простору

Технічні засоби читання-запису інформації (апаратна частина системи) мають справу з фізичною структурою дискового простору. Її описують у таких термінах:

- *диск* (disk) — фізичний пристрій для читання-запису інформації з автономним приводом;

- *робоча поверхня диску* (side) — магнітна поверхня диску, на якій зберігають дані. Вона пов’язана з *магнітною голівкою* (head) читання-запису;

- *магнітна доріжка* (track) — доріжка на робочій поверхні. Кожну доріжку поділено на сектори;

- *сектор* (sector) — мінімальна одиниця дискового простору, до котрої можна звернутися для запису чи читання інформації. Кожен сектор складається з *поля даних* та *поля службової інформації*, яка обмежує та ідентифікує його. У більшості ОС розмір сектора вибирають як 512 бай-

тів;

- *циліндр* (cylinder) — множина доріжок однакового радіусу, розташованих на всіх робочих поверхнях пакету дисків. Доступ до всіх доріжок одного циліндра можна забезпечити під час однократного радіального позиціювання блоку магнітних голівок. Сучасні жорсткі диски можуть мати декілька десятків тисяч циліндрів.

На рисунку 2.2 представлено фізичну структуру диску. Не варто плутати терміни «доріжка» й «циліндр», хоча на рисунку й може здатися, що це одне й те саме.

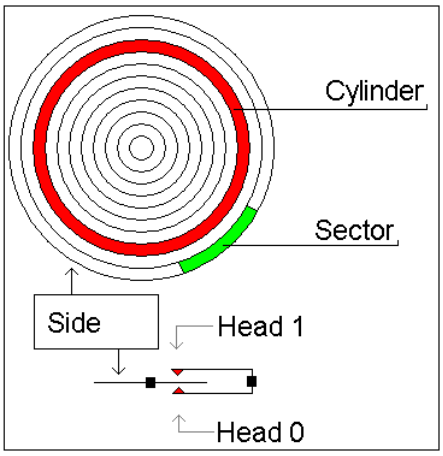

Рисунок 2.2 – Фізична структура диску

Усі робочі поверхні, доріжки й сектори послідовно пронумеровано. Фізичну адресу сектора на диску визначає тріада [c-h-s], де c — номер циліндра (cylinder), h — номер робочої поверхні диска (магнітної головки, head), s — номер сектора на доріжці (sector).

Для однозначного визначення місця розташування деякого файлу на диску потрібно зв’язати з цим файлом упорядковану послідовність пронумерованих секторів. Сектор — це доволі мала одиниця ємності дискового простору, тому його використання у якості логічної адреси файлу на диску

може створювати технічні труднощі та знижувати ефективність використання дисків великої ємності. Для подолання цих вад використовують крупнішу одиницю, яка має назву *кластер*. Кластер — це основний елемент логічної структури диску, що містить декілька секторів. Кількість секторів у кластері залежить від ємності диску та деяких інших параметрів.

2.2 Логічна структура дискового простору

Зазвичай, персональний комп'ютер комплектують одним чи декількома жорсткими дисками. Операційна система дозволяє розбивати диск на декілька частин, які система розглядатиме як окремі, *логічні*, диски. Розбивати жорсткий диск на логічні доцільно з декількох міркувань:

- у разі пошкодження логічного диску буде втрачено тільки ту інформацію, що містилася на цьому диску;

- реорганізація та вивантаження диску малого розміру виконується швидше;

- на одному диску може міститися декілька операційних систем, розташованих у різних розділах. У процесі початкового завантаження можна вказати розділ диску, із якого треба завантажувати потрібну операційну систему.

Логічна структура диску реалізує лінійну модель дискового простору. Згідно з цією моделлю, логічний диск розділено на дві послідовно розташовані області — системну й робочу.

Робочу область розташовано безпосередньо після системної. Її розділено на послідовно пронумеровані *кластери*. Призначення робочої обла-

сті — зберігання файлів та підкаталогів. Кластер використовують у якості мінімальної одиниці, яку виділяє ОС для одного файлу або підкаталогу. Наприклад, в ОС Windows (файлова система FAT32) кластер має розмір 32 КБ. Це означає, що якщо файл матиме розмір 1 КБ, для нього на диску все одно буде виділено 32 КБ. Кожен кластер має унікальний номер та містить декілька розташованих поспіль секторів (1 або 2 сектори для гнучких дисків, 4 та більше — для жорстких). Між номером кластера та списком абсолютних номерів секторів, що входять у нього, існує взаємно однозначне відображення.

Системною областю диску називають службову область, у якій не можуть зберігатися файли даних. Вона займає декілька початкових (в абсолютній нумерації) секторів, починаючи з 0-го, та містить такі блоки службової інформації:

- завантажувальний запис (Boot Record);
- кореневий каталог (Root Directory);
- таблиця розподілення файлів (File Allocation Table, FAT).

Ці блоки використовують для організації доступу до файлів та завантаження ОС. Ми розглянемо кожен із них детальніше в наступних розділах.

2.2.1 Головний завантажувальний запис

Найперший сектор жорсткого диску ([0-0-1]) містить так званий *головний завантажувальний запис* (Master Boot Record). Цей запис займає не весь сектор, а тільки його початкову частину. Головний завантажувальний запис являє собою програму, яку під час початкового завантаження

ОС із диску розміщують за адресою 7C00h:0000h, після чого їй передають керування. Завантажувальний запис продовжує процес завантаження ОС.

Оскільки всі логічні томи зберігають під загальною «обкладинкою» одного жорсткого диску, інформацію про розбиття останнього на томи фіксують наприкінці першого сектора жорсткого диску. У цій ділянці розташовано таблицю розділів диску (Partition Table), яка містить чотири елементи, що описують до чотирьох розділів диску. В останніх двох байтах сектора міститься значення 55AAh — ознака таблиці розділів. Розділи диску можуть бути активні або неактивні. Активний розділ можна використовувати для завантаження ОС. Жорсткий диск може мати декілька активних розділів, які можуть належать різним ОС.

Формат першого сектора жорсткого диску наведено в таблиці 2.1.

Таблиця 2.1 – Формат першого сектора жорсткого диску

Зміщення, Б	Розмір, Б	Опис
0	1BEh	Завантажувальний запис
1BEh	10h	Елемент таблиці розділів диску
1CEh	10h	Елемент таблиці розділів диску
1DEh	10h	Елемент таблиці розділів диску
1EEh	10h	Елемент таблиці розділів диску
1FEh	2	Ознака таблиці розділів (значення 55AAh)

Усі елементи таблиці розділів диску мають однаковий формат, наве-

дений у таблиці 2.2.

Таблиця 2.2 – Формат елементів таблиці розділів диску

Зміщення, Б	Розмір, Б	Опис
0	1	Ознака активного розділу (0 — неактивний; 80h — активний)
1	1	Номер голівки для початкового сектора розділу
2	2	Номер сектора й доріжки для початкового сектора розділу у форматі функції читання сектора `INT 13h`
4	1	Код системи (0 — невідома; 1, 4 — MS-DOS; 5 — розширений розділ MS-DOS)
5	1	Номер голівки для останнього сектора розділу
6	2	Номер сектора й доріжки для останнього сектора розділу у форматі функції читання сектора `INT 13h`
8	4	Відносний номер сектора початку розділу
12	4	Розмір розділу в секторах

2.2.2 Завантажувальний запис

У найпершому секторі активного розділу розташовано *завантажувальний запис* (Boot Record), який не варто плутати з головним завантажувальним записом. Головний завантажувальний запис зчитує завантажувальний запис в оперативну пам'ять, після чого останньому передають керування. Завантажувальний запис виконує завантаження ОС.

Окрім програми початкового завантаження ОС, завантажувальний запис містить параметри, що описують характеристики даного логічного диску. Ці параметри розташовано на початку сектора.

Формат завантажувального запису представлено в таблиці 2.3.

Таблиця 2.3 – Формат завантажувального запису

Зміщення, Б	Розмір, Б	Вміст
0	3	Команда JMP xxxx — ближній перехід на програму початкового завантаження
3	8	Назва фірми-виробника ОС та її версія
11	25	Extended BPB — розширений блок параметрів BIOS
36	1	Фізичний номер пристрою (0 — гнучкий диск, 80h — жорсткий диск)
37	1	Зарезервовано

Продовження таблиці 2.3

Зміщення, Б	Розмір, Б	Вміст
38	1	Символ ’)’ — ознака розширеного завантажувального запису
39	4	Серійний номер диску (Volume Serial Number), створюється під час форматування диску
43	11	Мітка диску (Volume Label)
54	8	Зарезервовано, зазвичай містить рядок, котрий ідентифікує формат FAT

На початку завантажувального сектора розташовано команду внутрішньосегментного переходу JMP. Вона потрібна для обходу форматованої зони сектора та передачі керування завантажувальній програмі, розташованій за зміщенням 62.

ОС не використовує назву фірми-виробника.

Поле зі зміщенням 38 завжди містить символ ‘)’, який означає, що використовується формат розширеного завантажувального запису.

Серійний номер диску формують під час форматування диску на основі дати й часу форматування. Це поле може бути використано для визначення факту заміни диску.

Мітку диску також формують під час форматування. Її можна замінити командою `LABEL` MS-DOS. Одночасно мітку диску поміщають у

кореневий каталог.

Поле завантажувального сектора зі зміщенням 11 містить *розширений блок параметрів BIOS* (Extended BPB). Його формат представлено в таблиці 2.4.

Таблиця 2.4 – Формат розширеного блоку параметрів BIOS

Зміщення, Б	Розмір, Б	Назва поля	Опис
0	2	`sectsize`	Кількість байтів в одному секторі диску
2	1	`clustsize`	Кількість секторів в одному кластері
3	2	`ressecs`	Кількість зарезервованих секторів
5	1	`fatcnt`	Кількість таблиць FAT
6	2	`rootsize`	Максимальна кількість дескрипторів файлів у кореневому каталозі диску
8	2	`totsecs`	Загальна кількість секторів на носії даних (у розділі MS-DOS)
10	1	`media`	Байт-опис середовища носія даних

Продовження таблиці 2.4

Зміщення, Б	Розмір, Б	Назва поля	Опис
11	2	`fatsize`	Кількість секторів, яку займає одна копія FAT
13	2	`seccnt`	Кількість секторів на доріжці
15	2	`headcnt`	Кількість магнітних голівок
17	2	`hiddensec_low`	Кількість прихованих секторів для розділу, котрий за розміром менший за 32 МБ
19	2	`hiddensec_hi`	Кількість прихованих секторів для розділу, котрий за розміром більший за 32 МБ
21	4	`drvsecs`	Загальна кількість секторів на логічному диску для розділу, котрий за розміром більший за 32 МБ

Для того, щоб отримати вміст завантажувального сектора, доцільно використовувати засоби звертання до секторів логічного диску за їхніми *логічними номерами*. MS-DOS організовує наскрізну нумерацію секторів, за якої кожному сектору логічного диску присвоєно свій (логічний) номер. Порядок нумерації обрано так, що за послідовного збільшення номера сектора спочатку збільшується номер голівки, а потім — номер доріжки. Це зроблено з метою мінімізації кількості переміщень блоку голівок під час звертання до послідовних логічних номерів секторів.

Приклад 2.1.

Нехай маємо дискету з 9 секторами на доріжці. Сектор із логічним номером 1 розташовано на 0-ій доріжці, для звертання до нього використовують 0-у голівку. Це — найперший сектор на доріжці.

Наступний сектор на 0-ій доріжці має логічний номер 2, останній — логічний номер 9. Сектор із логічним номером 10 також розташовано на 0-ій доріжці, і це також найперший сектор на доріжці, але для доступу до нього тепер використовують голівку з номером 1. І так далі, по мірі збільшення логічного номера сектора змінюються номери голівок і доріжок.

Для роботи з логічним диском на рівні логічних номерів секторів MS-DOS надає програмам два переривання — `INT 25h` (читання сектора за його логічним номером) та `INT 26h` (запис сектора за його логічним номером). Викликання цих переривань для версій MS-DOS, що підтримують розмір логічних дисків, більший за 32 МБ, має такий формат:

- на вході:

  `AL` — адреса диску (0 — A:, 1 — B: і т.д.);

  `CX` — завжди містить FFFFh (ознака того, що програма працює з

логічним диском, більшим за 32 МБ);

`DS:BX` — адреса структури, яка містить поля, представлені в таблиці 2.5;

- на виході:

`AH` — код помилки у випадку неуспішного завершення операції;

`CF` — ознака помилки (1, якщо помилка сталася, або 0, якщо помилки не було).

Таблиця 2.5 – Формат структури для роботи з логічними номерами секторів

Зміщення, Б	Розмір, Б	Уміст
0	4	Логічний номер початкового сектора
4	2	Кількість секторів для прочитання (переривання `INT 25h`) або запису (переривання `INT 26h`)
6	4	Дальня адреса буферу для передачі даних

Варто відмітити, що вказані переривання залишають у стеку одне слово — старий уміст регістру ознак. Тому після виклику переривання повинна слідувати команда `pop ax`.

Приклад 2.2.

Розгляньмо вихідний текст програми на мові програмування C, що показує вміст завантажувального запису для логічного диску, мітку якого вказує користувач (рисунок 2.3). Поле серійного номеру диску розбито на

дві компоненти — `volser_lo` та `volser_hi`. Це зроблено для спрощення представлення серійного номеру у тому вигляді, який використовує команда `DIR` MS-DOS.

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <ctype.h>

//структура для інформації з розширеного BPB
typedef struct _EBPB_
{
    unsigned sectsize;
    char clustsize;
    unsigned ressecs;
    char fatcnt;
    unsigned rootsize;
    unsigned totsecs;
    char media;
    unsigned fatsize;
    unsigned seccnt;
    unsigned headcnt;
    unsigned hiddensec_low;
    unsigned hiddensec_hi;
    unsigned long drvsecs;
} EBPB;

//структура для інформації з завантажувального запису
typedef struct _BOOT_
{
    char jmp[3];
    char oem[8];
    EBPB bpb;
    char drive;
    char reserved;
    char signature;
    unsigned volser_lo;
    unsigned volser_hi;
    char label[11];
    char fat_format[8];
    char boot_code[450];
} BOOT;

//структура для пари DS:BX
struct
{
    unsigned long first_sect;
    unsigned nsect;
    void far* buf;
} cb;

int getboot(BOOT far *boot, int drive);

int main(void)
{
```

```
char boot[512];
BOOT far* boot_rec = (BOOT far*)boot;
int i, status;
char drive;

//очищуємо екран
clrscr();

//запитуємо в користувача мітку тому
printf("\nPlease input the disc volume: ");
drive = getche();

//отримуємо номер тому
drive = toupper(drive) - 'A';

//отримуємо завантажувальний запис
status = getboot((BOOT far*) boot_rec, drive);

if(status)
{
    //якщо сталася помилка
    printf("\nError while reading the boot sector!");
    return -1;
}

//виводимо інформацію з завантажувального сектора
printf("\nBoot sector for disc %c", drive + 'A');
printf("\n"
    "\nOEM - company name and DOS version: ");

for(i = 0; i < 8; i++)
{
    printf("%c", boot_rec->oem[i]);
}

printf("\nVolume number: %x"
    "\nExtended BOOT record flag: %c"
    "\nVolume serial number: %04X-%04X"
    "\nVolume label: ",
    (unsigned char)boot_rec->drive,
    boot_rec->signature,
    boot_rec->volser_hi,
    boot_rec->volser_lo);

for(i = 0; i < 11; i++)
{
    printf("%c", boot_rec->label[i]);
}

printf("\nFAT format: ");
for(i = 0; i < 8; i++)
{
    printf("%c", boot_rec->fat_format[i]);
}

printf("\n\nBPB info:");
printf("\nBytes in a sector: %d"
```

```
        "\nSectors in a cluster: %d"
        "\nSectors reserved: %d"
        "\nFAT copies number: %d"
        "\nMax files in a root directory: %d"
        "\nTotal sectors on a disc: %d"
        "\nMedia descriptor byte: %x"
        "\nSectors in FAT: %d",
        boot_rec->bpb.sectsize, boot_rec->bpb.clustsize,
        boot_rec->bpb.ressecs,  boot_rec->bpb.fatcnt,
        boot_rec->bpb.rootsize, boot_rec->bpb.totsecs,
        (unsigned char) boot_rec->bpb.media,
        boot_rec->bpb.fatsize);

        printf("\n\nExtended BPB info:");
        printf("\nSectors on a track: %d"
        "\nHead count: %d"
        "\nHidden sectors for a disc < 32MB: %d"
        "\nHidden sectors for a disc >= 32MB: %d"
        "\nTotal sectors on a disc: %u",
        boot_rec->bpb.seccnt, boot_rec->bpb.headcnt,
        boot_rec->bpb.hiddensec_low,
        boot_rec->bpb.hiddensec_hi,
        boot_rec->bpb.totsecs);

    return 0;
}

//функція для зчитування інформації з завантажувального сектора
int getboot(BOOT far *boot, int drive)
{
    cb.first_sect = 0;
    cb.nsect = 1;
    cb.buf = (void far*)boot;

    _BX = FP_OFF(&cb);
    _DS = FP_SEG(&cb);
    _CX = 0xffff;
    _DX = 0;
    _AX = drive;
    asm int 25h
    asm pop ax
    asm jc err

    return 0;
err:
    return 1;
}
```

У вищенаведеній програмі використано макрокоманди FP_SEG та FP_OFF, описані у файлі DOS.H. Ці макрокоманди дозволяють виділити з дальнього вказівника сегмент та зміщення, відповідно.

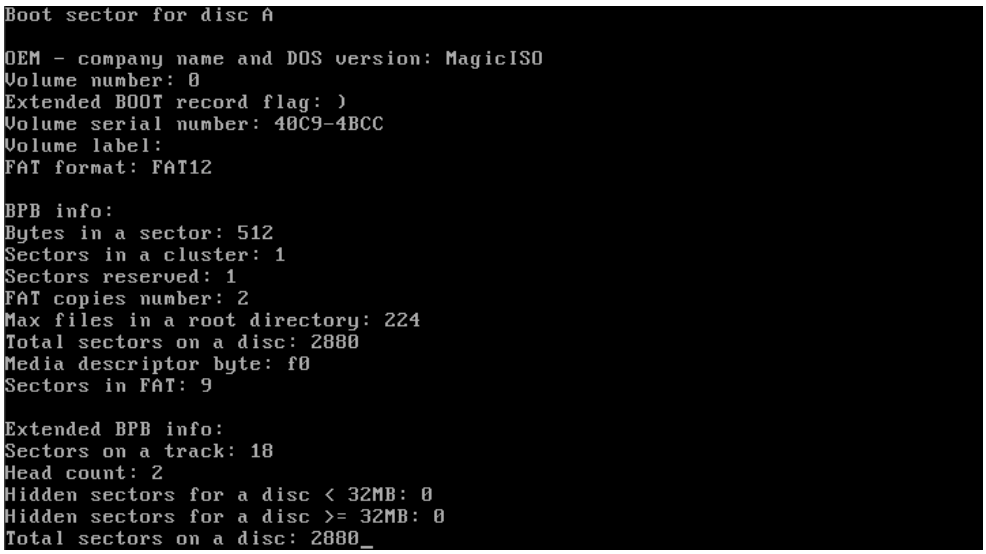

Рисунок 2.3 – Результат виконання програми з Прикладу 2.2

2.2.3 Таблиця розподілення файлів

Відразу за завантажувальним сектором на логічному диску містяться сектори, які складають *таблицю розподілення файлів* (File Allocation Table, FAT).

Операційна система MS-DOS використовує дисковий простір у такий спосіб. Під час створення файлу для нього не здійснюють початкове розподілення пам’яті доріжками чи секторами. По мірі збільшення файлу в розмірах, ОС розподіляє цьому файлу сектори з числа вільних (тих, що не використовують інші файли). При цьому файл не обов’язково розташований у суміжних ділянках диску, і його може бути розкидано різними доріжками та секторами.

Очевидно, що в цьому випадку ОС повинна відслідковувати використовувані сектори диску та зберігати для кожного файлу інформацію про виділені йому сектори. В ОС MS-DOS для цих цілей і використовують FAT.

FAT — це масив, який містить інформацію про кластери диску. Для кожного кластера у FAT є окрема комірка, у котрій зберігають інформацію про його використання. Розмір FAT визначає загальна кількість кластерів на логічному диску.

Якщо файл займає декілька кластерів, то їх зв’язано в список. При цьому елементи FAT містять номери наступних використовуваних даним файлом кластерів. Кінець списку відмічено спеціальним значенням. Усі вільні кластери у FAT відмічено нулями.

Номер першого кластера, розподіленого файлу, зберігається в елементі каталогу, що описує даний файл. Ми розглядатимемо структуру каталогу детальніше в наступному розділі.

Приклад 2.3.

На рисунку 2.4 зображено приклад розподілення кластерів для файлів `AUTOEXEC.BAT` та `CONFIG.SYS`. На рисунку можна бачити, що файлу `AUTOEXEC.BAT` розподілено три кластери, а файлу `CONFIG.SYS` — два.

Також можна бачити, що в каталозі, окрім іншої інформації, вказано номери перших кластерів, розподілених цим файлів (11 та 12, відповідно). В 11-ій комірці FAT містить число 17, тобто номер другого кластера, розподіленого файлу `AUTOEXEC.BAT`. Комірка з номером 17, у свою чергу, містить число 18 — номер третього кластера цього файлу. Остання комірка, яка відповідає останньому кластеру цього файлу, містить спеціальне значення — FFFFh.

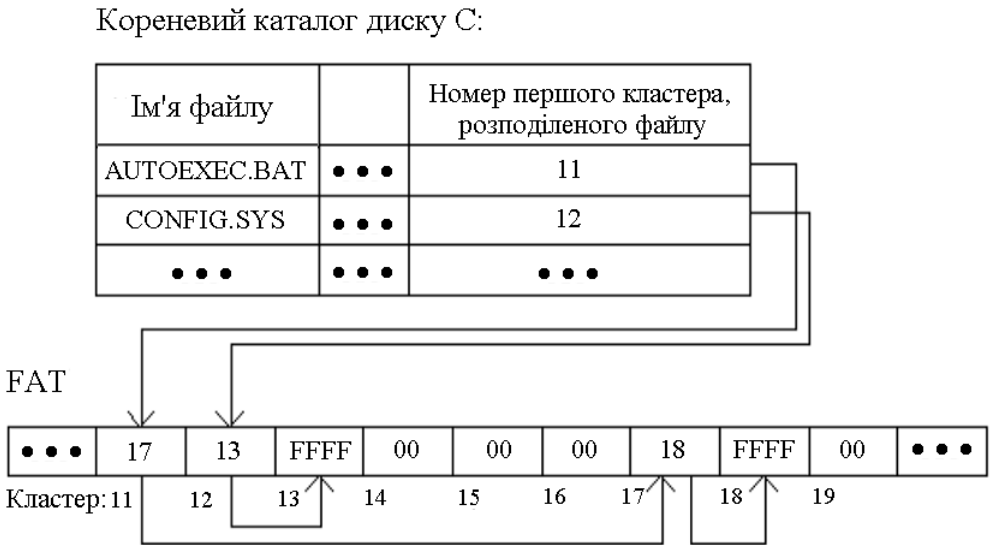

Рисунок 2.4 – Фрагмент FAT із Прикладу 2.3

FAT може бути двох форматів:

- 12-бітовий: для збереження інформації про один кластер використовують 12 бітів. Цей формат зручний для дискет із невеликою кількістю секторів: уся FAT займає один сектор;

- 16-бітовий: для збереження інформації про один кластер використовують 16 бітів. Цей формат використовують для дисків розміром, більшим за 32МБ.

Визначити формат FAT можна шляхом аналізу рядка довжиною 8 байтів, який розміщено за зміщенням 54 в завантажувальному записі (таблиця 2.3). Цей рядок може містити значення `"FAT12"` або `"FAT16"`, залежно від формату FAT. У програмі з Прикладу 2.2 цей рядок мав назву `fat_format`.

Перший байт FAT — опис середовища (Media Descriptor), який має таке саме значення, як і байт-опис середовища із завантажувального запису. Наступні 5 байтів (7 байтів) для 12-бітового (16-бітового) формату FAT завжди містять значення FFh.

Решта таблиці містить 12- або 16-бітові комірки (залежно від формату FAT), кожна з яких відповідає певному кластеру та може набувати значень, представлених у таблиці 2.6.

Таблиця 2.6 – Можливі значення комірок FAT

FAT12	FAT16	Інтерпретація
000h	0000h	Вільний кластер
FF0h–FF6h	FFF0h–FFF6h	Зарезервований кластер
FF7h	FFF7h	Пошкоджений кластер
FF8h–FFFh	FFF8h–FFFFh	Останній кластер у списку
002h–FEFh	0002h–FFEFh	Номер наступного кластера в списку

Розгляньмо алгоритм прочитання інформації з FAT. Безпосередній доступ до FAT може бути доцільний для організації сканування каталогів під

час пошуку файлів, для прочитання каталогів як файлів, для організації захисту інформації від несанкційованого копіювання тощо. Загальний алгоритм використання FAT передбачає виконання таких кроків:

а) зчитати FAT у пам'ять. Для визначення початкового сектора FAT потрібно зчитати в пам'ять завантажувальний сектор та проаналізувати вміст BPB: поле `ressecs` містить кількість зарезервованих кластерів, які знаходяться перед FAT. Окрім цього, поле `fatsize` містить розмір FAT у секторах.

Також варто враховувати, що на диску може міститися декілька копій FAT (ОС використовує тільки першу копію, але оновлює другу; інші копії потрібні для утиліт відновлення вмісту диску). Кількість копій FAT міститься у полі `fatcnt` завантажувального сектора;

б) отримати номер першого кластера файлу, для якого потрібно визначити список розподілених кластерів;

в) використати цей номер як індекс у FAT для отримання номеру наступного кластера;

г) повторювати цю процедуру доти, доки значення з FAT не буде відповідати кінцю файлу (таблиця 2.6).

Крок 3 із наведеного вище алгоритму різниться залежно від формату FAT. Так, якщо маємо FAT16, то для визначення номеру наступного кластера достатньо просто отримати 16-бітове слово з FAT, використавши у якості індексу номер попереднього кластера.

Для FAT12 процедура визначення номеру наступного кластера передбачає виконання таких кроків:

а) помножити номер початкового кластера на 3 та розділити на 2 (кожен елемент таблиці має довжину 1,5 байта);

б) зчитати 16-бітове слово з FAT, використовуючи в якості зміщення отримане вище значення;

в) якщо номер початкового кластера парний, на вибране з FAT слово потрібно накласти маску 0FFFh (залишивши 12 молодших бітів); інакше — зсунути вправо вибране з FAT слово на 4 біти (залишивши старші 12 бітів);

г) отриманий результат являтиме собою номер наступного кластера в списку, або ознаку кінця списку (таблиця 2.6).

2.2.4 Кореневий каталог

Як було розглянуто в розділі 2.1.1, файлова система MS-DOS має деревоподібну структуру, у корені якої розташовано *кореневий каталог* (Root Directory). У кореневому каталозі (як і в будь-якому іншому) містяться 32-байтові *дескриптори*, які містять інформацію про файли та підкаталоги. Таким чином, для того, щоб отримати безпосередній доступ до файлів та підкаталогів, потрібно вміти визначати розташування та розмір кореневого каталогу.

Кореневий каталог міститься відразу за останньою копією FAT. Ураховуючи те, що кількість секторів, які займає одна копія FAT, міститься в полі `fatsize` BPB, кількість копій FAT — у полі `fatcnt`, а кількість зарезервованих секторів — у полі `ressecs`, то перед кореневим каталогом розташовано 1 завантажувальний сектор, `ressecs` зарезервованих секторів та $\texttt{fatcnt} \cdot \texttt{fatsize}$ секторів FAT.

Кореневий каталог займає неперервну область фіксованого розміру, який можна визначити, маючи значення поля `rootsize` BPB. Під час форматування диску в це поле записують максимальне число файлів та підкаталогів, яке може міститися в кореневому каталозі. Оскільки для кожного елемента каталогу відведено 32 байти, кореневий каталог має довжину $32 \cdot$ `rootsize` байтів.

Розгляньмо тепер структуру дескриптора. Його формат наведено в таблиці 2.7.

Таблиця 2.7 – Структура дескриптора

Зміщення, Б	Розмір, Б	Вміст
0	8	Ім’я файлу (каталогу), вирівняне за лівою границею та доповнене пробілами
8	3	Розширення імені файлу, вирівняне за лівою границею та доповнене пробілами

Продовження таблиці 2.7

Зміщення, Б	Розмір, Б	Вміст
11	1	Байт атрибутів файлу
12	10	Зарезервовано
22	2	Час створення файлу або час його останньої модифікації
24	2	Дата створення файлу або дата його останньої модифікації
26	2	Номер першого кластера, розподіленого файлу
28	4	Розмір файлу в байтах

Біти байту атрибутів файлу мають такі значення:

- 0: файл тільки для читання;
- 1: прихований файл;
- 2: системний файл;
- 3: мітка диску;
- 4: підкаталог даного каталогу;
- 5: ознака архівації: якщо дорівнює 1, то файл не було вивантажено утилітою архівації;
- 6–7: зарезервовано.

Час створення файлу має наступний формат:

- старші 5 бітів цього поля містять значення години модифікації файлу;
- 6 бітів із номерами 5–10 містять значення хвилин модифікації файлу;
- молодші 5 бітів містять значення секунд модифікації файлу, поділене на 2.

Дата створення файлу має наступний формат:

- старші 7 бітів цього поля містять значення, що дорівнює результату віднімання від року модифікації файлу числа 1980;
- 4 біти з номерами 5–8 містять значення місяця модифікації файлу;
- молодші 5 бітів містять значення дня модифікації файлу.

Варто окремо зазначити, що під час видалення файлу або підкаталогу відповідний дескриптор не видаляють: замість цього перший символ імені замінюють на байт із номером 229 (символ `x` у кодуванні CP866).

У кореневому каталозі перші два дескриптори відповідають системний файлам ОС IO.SYS та MSDOS.SYS. У будь-якому каталозі, окрім кореневого, перші два дескриптори мають спеціальне призначення:

- перший дескриптор містить у полі імені рядок "`.  `". Цей дескриптор указує на каталог, який його містить, тобто каталог має посилання сам на себе;
- другий дескриптор містить у полі імені рядок "`..  `". Цей дескриптор указує на батьківський каталог.

Якщо в полі номера першого кластера для дескриптора з іменем "`..`" стоїть 0, це значить, що відповідний каталог міститься в кореневому каталозі.

На рисунку 2.5 представлено всі області логічного диску. Таку структуру, окрім логічних дисків, також мають дискети.

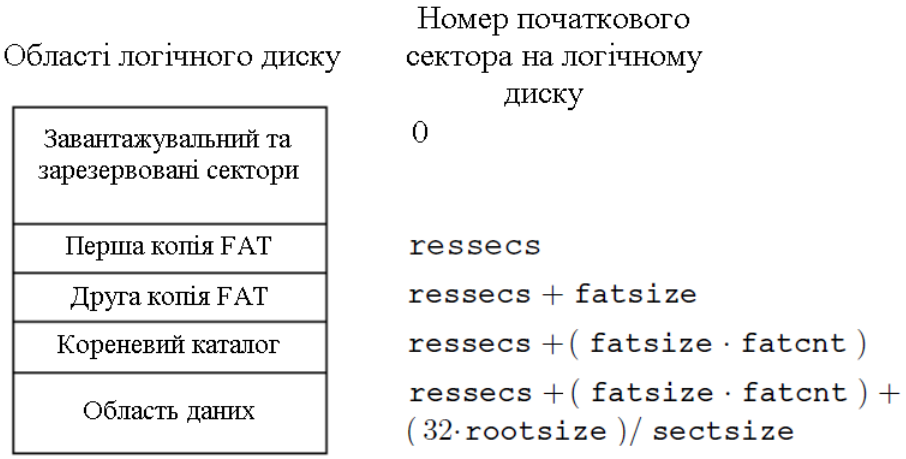

Рисунок 2.5 – Структура логічного диску MS-DOS

Область даних розбито на кластери, при цьому нумерація кластерів починається з 2: кластеру номер 2 відповідають перші сектори області даних. На основі вказаної на рисунку 2.5 інформації можна з легкістю обчислити логічний номер першого сектора будь-якого кластера з області даних.

Приклад 2.4.

Розгляньмо вихідний текст програми на мові програмування С, що зчитує завантажувальний сектор вибраного диску, обчислює розмір і розташування кореневого каталогу та виводить на екран його вміст (рисунок 2.6).

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <dos.h>
#include <ctype.h>

//структура для інформації з розширеного BPB
typedef struct _EBPB_
{
    unsigned sectsize;
    char clustsize;
    unsigned ressecs;
    char fatcnt;
    unsigned rootsize;
    unsigned totsecs;
    char media;
    unsigned fatsize;
    unsigned seccnt;
    unsigned headcnt;
    unsigned hiddensec_low;
    unsigned hiddensec_hi;
    unsigned long drvsecs;
} EBPB;

//структура для інформації з завантажувального запису
typedef struct _BOOT_
{
    char jmp[3];
    char oem[8];
    EBPB bpb;
    char drive;
    char reserved;
    char signature;
    unsigned volser_lo;
    unsigned volser_hi;
    char label[11];
    char fat_format[8];
    char boot_code[450];
} BOOT;
```

```
//структура для часу створення файлу
typedef struct _FTIME_
{
    unsigned sec : 5, min : 6, hour : 5;
} FTIME;

//структура для дати створення файлу
typedef struct _FDATE_
{
    unsigned day : 5, month : 4, year : 7;
} FDATE;

//структура для дескриптора файлу
typedef struct _FITEM_
{
    char name[8];
    char ext[3];
    char attr;
    char reserved[10];
    FTIME time;
    FDATE date;
    unsigned cluster_nu;
    unsigned long size;
} FITEM;

//структура для пари DS:BX
struct
{
    unsigned long first_sect;
    unsigned nsect;
    void far* buf;
} cb;

int getboot(BOOT far *boot, int drive);

int main(void)
{
    int i, status;
    int root_begin, root_sectors;
    char drive;

    char boot[512];
    BOOT far* boot_rec = (BOOT far*) boot;

    FITEM *root_buffer, far *rptr;

    //очищуємо екран
    clrscr();

    //запитуємо в користувача мітку тому
    printf("\nPlease input the volume label: ");
    drive = getche();

    //отримуємо номер тому
    drive = toupper(drive) - 'A';

    //отримуємо завантажувальний запис
```

```
status = getboot((BOOT far*)boot_rec, drive);

if(status)
{
    //якщо сталася помилка
    printf("\nError while reading the boot sector!");
    return -1;
}

//отримуємо перший сектор кореневого каталогу
root_begin = boot_rec->bpb.ressecs +
    boot_rec->bpb.fatsize * boot_rec->bpb.fatcnt;

//отримуємо розмір кореневого каталогу
root_sectors = (boot_rec->bpb.rootsize * 32) /
    boot_rec->bpb.sectsize;

//виділяємо пам'ять для кореневого каталогу
root_buffer = (FITEM *)
    malloc(root_sectors * boot_rec->bpb.sectsize);

if(root_buffer == NULL)
{
    //якщо сталася помилка
    printf("\nNot enough memory!");
    return -1;
}

//зчитуємо кореневий каталог у виділений буфер
cb.first_sect = root_begin;
cb.nsect = root_sectors;
cb.buf = (void far*) root_buffer;

_BX = FP_OFF(&cb);
_DS = FP_SEG(&cb);
_CX = 0xffff;
_DX = 0;
_AX = drive;
asm int 25h
asm pop ax
asm jc error

//виводимо інформацію з кореневого каталогу
printf("\n"
    "\nFile name        Attr. Date                "
    "Time            Cluster    Size    "
    "\n------------ ----- ----------    "
    "--------    ------- ------");

for(rptr = root_buffer; ; rptr++)
{
    printf("\n");

    if(rptr->name[0] == 0 ||
         rptr->name[0] == (char) 0xF6)
    {
        //кінець каталогу: нульовий байт на початку
```

```
            //імені файлу або байт 0xF6 (порожній каталог)
            break;
        }

        for(i = 0; i < 8; i++)
        {
            printf("%c", rptr->name[i]);
        }

        printf(".");

        for(i = 0; i < 3; i++)
        {
            printf("%c", rptr->ext[i]);
        }

        printf(" %02X        %02d-%02d-%02d    %02d:%02d:%02d ",
            rptr->attr, rptr->date.day,
            rptr->date.month, rptr->date.year + 1980,
            rptr->time.hour, rptr->time.min,
            rptr->time.sec * 2);

        printf(" %-6ld     %lu",
            (long) rptr->cluster_nu, (long) rptr->size);
    }

    //очищуємо пам'ять
    free(root_buffer);

    return 0;
error:
    //якщо сталася помилка
    printf("\nError while reading the directory!");

    //очищуємо пам'ять
    free(root_buffer);

    return -1;
}

//функція для зчитування інформації з завантажувального сектора
int getboot(BOOT far *boot, int drive)
{
    cb.first_sect = 0;
    cb.nsect = 1;
    cb.buf = (void far*) boot;

    _BX = FP_OFF(&cb);
    _DS = FP_SEG(&cb);
    _CX = 0xffff;
    _DX = 0;
    _AX = drive;

    asm int 25h

    asm pop ax
    asm jc err
```

```
    return 0;
err:
    return 1;
}
```

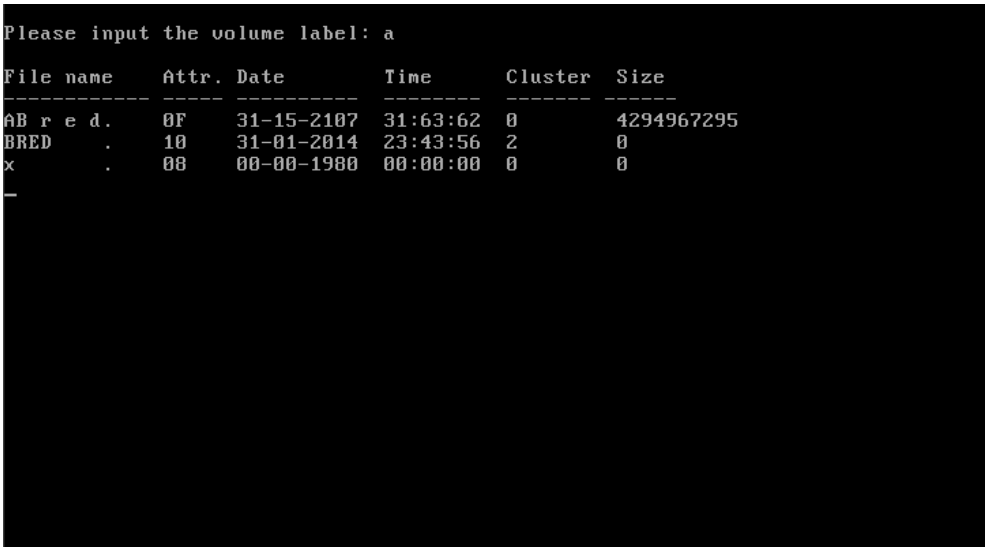

Рисунок 2.6 – Результат виконання програми з Прикладу 2.4

Приклад 2.5.

Розгляньмо вихідний текст програми на мові програмування С, що демонструє основні можливості роботи з FAT. Програму призначено для роботи з гнучкими дисками. У процесі роботи програма спочатку виводить на екран уміст кореневого каталогу вибраного користувачем гнучного диску, після чого запитує в користувача номер початкового кластера деякого підкаталогу. Отримавши від користувача номер кластера, програма виводить уміст цього каталогу кластер за кластером (рисунок 2.7).

```
//Для цієї програми потрібна модель пам'яті Large

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <malloc.h>
#include <dos.h>
#include <bios.h>
#include <ctype.h>
#include <string.h>
```

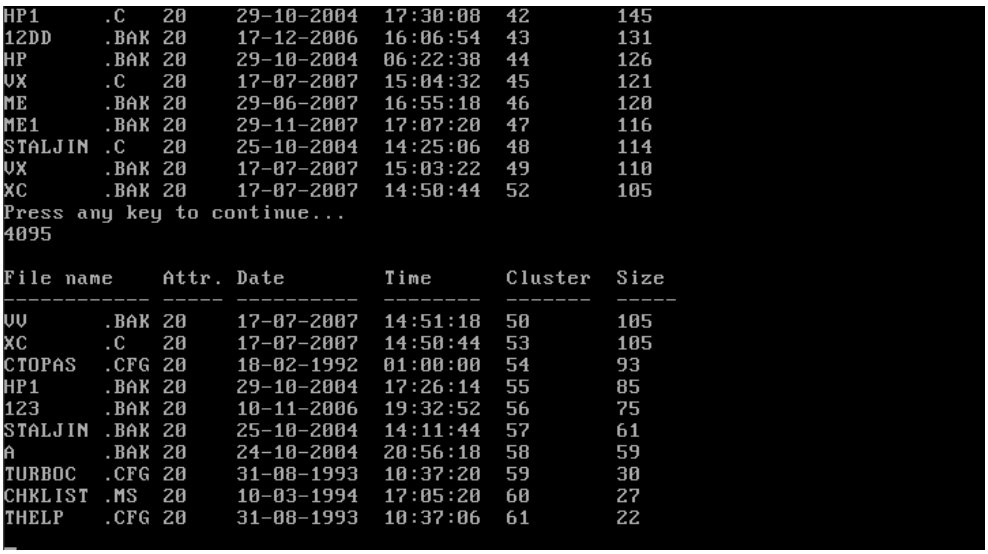

Рисунок 2.7 – Результат виконання програми з Прикладу 2.5

```
//структура для інформації з розширеного BPB
typedef struct _EBPB_
{
    unsigned sectsize;
    char clustsize;
    unsigned ressecs;
    char fatcnt;
    unsigned rootsize;
    unsigned totsecs;
    char media;
    unsigned fatsize;
    unsigned seccnt;
    unsigned headcnt;
    unsigned hiddensec_low;
    unsigned hiddensec_hi;
    unsigned long drvsecs;
} EBPB;

//структура для інформації з завантажувального запису
typedef struct _BOOT_
{
    char jmp[3];
    char oem[8];
    EBPB bpb;
    char drive;
    char reserved;
    char signature;
    unsigned volser_lo;
    unsigned volser_hi;
    char label[11];
    char fat_format[8];
    char boot_code[450];
} BOOT;

//структура для часу створення файлу
```

```
typedef struct _FTIME_
{
    unsigned sec : 5, min : 6, hour : 5;
} FTIME;

//структура для дати створення ƒайлу
typedef struct _FDATE_
{
    unsigned day : 5, month : 4, year : 7;
} FDATE;

//структура для дескриптора ƒайлу
typedef struct _FITEM_
{
    char name[8];
    char ext[3];
    char attr;
    char reserved[10];
    FTIME time;
    FDATE date;
    unsigned cluster_nu;
    unsigned long size;
} FITEM;

//структура для пари DS:BX
struct
{
    unsigned long first_sect;
    unsigned nsect;
    void far* buf;
} cb;

int getboot(BOOT far *boot, int drive);
int fat(char far *b_fat, int t_fat, unsigned idx);

int main(void)
{
    BOOT *boot_rec;
    int i,j, k, status, fat_sectors;
    unsigned total_sectors;
    int ffat, root_begin, root_sectors;
    char drive;
    unsigned far *fat_buffer;
    FITEM far *root_buffer, far *rptr;
    char cbuf[128];
    char far *clust_buffer;
    unsigned cur_clust;

    //очищуємо екран
    clrscr();

    //виділямо пам'ять для завантажувального запису
    boot_rec = (BOOT*) malloc(sizeof(BOOT));
    if(boot_rec == NULL)
    {
        //якщо сталася помилка
        printf("\nNot enough memory!");
```

```
    return -1;
}

while(1)
{
    //запитуємо в користувача мітку тому
    printf("\nPlease input the volume label: ");
    drive = getche();

    //отримуємо номер тому
    drive = toupper(drive) - 'A';

    if(drive > 1)
    {
        //якщо сталася помилка
        printf("\nOnly floppies are allowed!\n");
        continue;
    }
    else
    {
        break;
    }
}

//отримуємо завантажувальний запис
status = getboot((BOOT _far*)boot_rec, drive);
if(status)
{
    //якщо сталася помилка
    printf("\nError while reading the boot sector!");

    //очищуємо пам'ять
    free(boot_rec);

    return -1;
}

//отримуємо загальне число секторів
total_sectors = boot_rec->bpb.totsecs;

if(total_sectors == 0){
    //якщо в нас розширений BPB,
    //то треба використати інше поле
    total_sectors = boot_rec->bpb.drvsecs;
}

//визначаємо формат FAT
if(!strncmp(boot_rec->fat_format, "FAT16", 5))
{
    printf("\nFAT has 16 bit format!");
    ffat = 16;
}
else
{
    printf("\nFAT has 12 bit format!");
    ffat = 12;
}
```

```
//отримуємо число секторів у FAT
fat_sectors = boot_rec->bpb.fatsize;

//виділяємо пам'ять для FAT
fat_buffer = (unsigned int*)
    farmalloc((long) fat_sectors * boot_rec->bpb.sectsize);
if(fat_buffer == NULL)
{
    //якщо сталася помилка
    printf("\nNot enough memory!");

    //очищуємо пам'ять
    free(boot_rec);

    return -1;
}

//отримуємо перший сектор FAT
j = boot_rec->bpb.ressecs;

//зчитуємо FAT у fat_buffer
cb.first_sect = j;
cb.nsect = fat_sectors;
cb.buf = (void far*)fat_buffer;
_BX = FP_OFF(&cb);
_DS = FP_SEG(&cb);
_CX = 0xffff;
_DX = 0;
_AX = drive;
asm int 25h
asm pop ax

//отримуємо перший сектор кореневого каталогу
root_begin = j + fat_sectors * boot_rec->bpb.fatcnt;

//отримуємо розмір кореневого каталогу
root_sectors = (boot_rec->bpb.rootsize * 32) /
    boot_rec->bpb.sectsize;

//виділяємо пам'ять для кореневого каталогу
root_buffer = (FITEM far *)
    farmalloc(root_sectors * boot_rec->bpb.sectsize);
if(root_buffer == NULL)
{
    //якщо сталася помилка
    printf("\nNot enough memory!");

    //очищуємо пам'ять
    free(boot_rec);
    farfree(fat_buffer);

    return -1;
}

//зчитуємо кореневий каталог у виділений буфер
cb.first_sect = root_begin;
```

```
cb.nsect = root_sectors;
cb.buf = (void far*) root_buffer;

_BX = FP_OFF(&cb);
_DS = FP_SEG(&cb);
_CX = 0xffff;
_DX = 0;
_AX = drive;
asm int 25h
asm pop ax

//виводимо інформацію з кореневого каталогу
printf("\n"
    "\nFile name        Attr. Date                "
    "Time            Cluster    Size    "
    "\n------------ ----- ----------    "
    "--------    ------- ------");

for(rptr = root_buffer; ;rptr++)
{
    printf("\n");

    if(rptr->name[0] == 0)
    {
        //кінець каталогу: нульовий байт
        //на початку імені файлу
        break;
    }

    for(i = 0; i < 8; i++)
    {
        printf("%c", rptr->name[i]);
    }

    printf(".");

    for(i = 0; i < 3; i++)
    {
        printf("%c", rptr->ext[i]);
    }

    printf(" %02X        %02d-%02d-%02d    %02d:%02d:%02d ",
        rptr->attr, rptr->date.day,
        rptr->date.month, rptr->date.year + 1980,
        rptr->time.hour, rptr->time.min,
        rptr->time.sec * 2);

    printf(" %-6ld     %lu",
        (long) rptr->cluster_nu, (long) rptr->size);
}

//виділямо пам'ять для кластерів каталогу
clust_buffer = (char far*)
    farmalloc(boot_rec->bpb.clustsize
    * boot_rec->bpb.sectsize);
if(clust_buffer == NULL)
{
```

```
    //якщо сталася помилка
    printf("\nNot enough memory!");

    //очищуємо пам'ять
    farfree(root_buffer);
    free(boot_rec);
    farfree(fat_buffer);

    return -1;
}

printf("\nFirst directory cluster: ");

//запитуємо в користувача номер кластера
gets(cbuf);
cur_clust = atol(cbuf);

k = 0;

for(;;)
{
    //записуємо номер кластера каталогу
    unsigned long j = cur_clust;

    //отримуємо наступний кластер
    cur_clust = fat((char far*) fat_buffer, ffat, cur_clust);
    printf("\n%d ", cur_clust);

    //зчитуємо цей кластер у clust_buffer
    cb.first_sect = root_begin + root_sectors
        + ((j - 2) * boot_rec->bpb.clustsize);
    cb.nsect = boot_rec->bpb.clustsize;
    cb.buf = (void far*) clust_buffer;
    _BX = FP_OFF(&cb);
    _DS = FP_SEG(&cb);
    _CX = 0xffff;
    _DX = 0;
    _AX = drive;
    asm int 25h
    asm pop ax

    //виводимо вміст каталогу
    rptr = (FITEM far *) clust_buffer;

    if(k == 0)
    {
        k = 1;
        if(_fstrncmp(rptr->name,".             ",8) != 0)
        {
            //якщо це не каталог (бо не містить
            //дескриптора ".")
            printf("\nThis is not a directory!");

            //очищуємо пам'ять
            farfree(root_buffer);
            free(boot_rec);
            farfree(fat_buffer);
```

```
        farfree(clust_buffer);

        return -1;
    }
}

printf("\n"
    "\nFile name        Attr. Date               "
    "Time            Cluster    Size    "
    "\n------------ ----- ----------    "
    "--------    -------    -----");

int counter = 0;
for(;; rptr++)
{
    counter++;
    printf("\n");

    if(rptr->name[0] == 0)
    {
        //кінець каталогу: нульовий байт
        //на початку імені файлу
        break;
    }

    //можемо прочитати тільки (clustsize * sectsize) / 32 дескрипторів
    if (counter > (boot_rec->bpb.clustsize * boot_rec->bpb.sectsize) / 32)
    {
        break;
    }

    for(i = 0; i < 8; i++)
    {
        printf("%c", rptr->name[i]);
    }

    printf(".");

    for(i = 0; i < 3; i++)
    {
        printf("%c", rptr->ext[i]);
    }

    printf(" %02X        %02d-%02d-%02d    %02d:%02d:%02d ",
        rptr->attr, rptr->date.day,
        rptr->date.month, rptr->date.year + 1980,
        rptr->time.hour, rptr->time.min,
        rptr->time.sec * 2);

    printf(" %-6ld     %lu",
        (long) rptr->cluster_nu, (long) rptr->size);
}

//якщо цей кластер --- останній
//серед кластерів каталогу, виходимо
if((cur_clust >= 0xff8 && cur_clust <= 0xfff)
    ||
```

```
        (cur_clust >= 0xfff8 && cur_clust <= 0xffff)) break;

        printf("Press any key to continue...");
        getch();
    }

    //очищуємо пам'ять
    farfree(root_buffer);
    free(boot_rec);
    farfree(fat_buffer);
    farfree(clust_buffer);

    return 0;
}

//функція для зчитування інформації з завантажувального сектора
int getboot(BOOT far *boot, int drive)
{
    cb.first_sect = 0;
    cb.nsect = 1;
    cb.buf = (void far*)boot;

    _BX = FP_OFF(&cb);
    _DS = FP_SEG(&cb);
    _CX = 0xffff;
    _DX = 0;
    _AX = drive;
    asm int 25h
    asm pop ax
    asm jc err

    return 0;
err:
    return 1;
}

//функція для обходу кластерів у FAT
int fat(char far *b_fat, int t_fat, unsigned idx)
{
    div_t clust_nu;
    int cluster;

    if(t_fat == 12)
    {
        clust_nu = div(idx * 3, 2);

        if(clust_nu.rem != 0)
             cluster =
                 (*((int far*) (b_fat +
                 clust_nu.quot)) >> 4) & 0xfff;
        else
             cluster =
                 *((int far*) (b_fat + clust_nu.quot)) & 0xfff;
    }
    else if(t_fat == 16)
    {
        cluster = *((int far*) (b_fat + idx * 2));
```

```
    }
    else
    {
        //якщо сталася помилка
        printf("Error in the FAT format!\n");
        exit(-100);
    }

    return cluster;
}
```

3 ПОРЯДОК ЗДАЧІ ЛАБОРАТОРНОЇ РОБОТИ ТА ВИМОГИ ДО ЗВІТУ

Здача лабораторної роботи передбачає:

- демонстрацію працездатності розробленої в рамках роботи програми в ОС MS-DOS 6.22;
- здачу викладачеві звіту з лабораторної роботи.

Під час здавання лабораторної роботи викладач має право ставити студентові питання за матеріалами розділу 2.

Під час демонстрації роботи програми викладач має право вимагати від студента внесення змін у програму.

Звіт про виконання лабораторної роботи повинен включати:

а) вступ;

б) постановку задачі;

в) теоретичні відомості;

г) опис розробленої програми;

д) результати випробування розробленої програми;

е) висновки;

є) посилання на літературні джерела;

ж) додаток, у якому повинно бути розміщено лістинг розробленої програми.

Детальніші вимоги до звіту викладено в документі «Вимоги до оформлення звіту».

Текстову частину повинно бути оформлено відповідно до вимог ДСТУ 3008-95 «Документація. Звіти у сфері науки і техніки. Структура і правила оформлення» із застосуванням системи підготовки документів LaTeX.

ПЕРЕЛІК ПОСИЛАНЬ

1. Фролов А. MS-DOS для программиста / А. Фролов, Г. Фролов. — [Том 19] — М. : Диалог-МИФИ, 1995. — 253 с.

2. Duncan R. Advanced MS-DOS Programming: The Microsoft Guide for Assembly Language and C Programmers / R. Duncan. — Microsoft Press, 1988. — 686 p.

3. Abel P. IBM PC Assembly Language and Programming / P. Abel. — [5th ed.] — Prentice Hall, 2001. — 540 с.