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

Кафедра прикладної математики

ДОСЛІДЖЕННЯ ВЕКТОРНОЇ ТАБЛИЦІ ЗВ'ЯЗКІВ В MS-DOS

методичні рекомендації до виконання лабораторної роботи №2

з дисципліни:

«Операційні системи»

Київ — 2015

Зміст

ВСТУП

Операційна система (ОС) MS-DOS (Microsoft Disk Operating System) містить векторну таблицю зв’язків основних блоків керування. Маючи адресу цієї таблиці, можна отримати доступ до внутрішніх структур даних операційної системи.

На початку оперативної пам’яті розміщено область даних BIOS. Після цієї області розміщено область даних MS-DOS, яка містить внутрішні змінні та структури даних ОС. Основні структури даних організовано у вигляді дерева, коренем якої є векторна таблиця зв’язків. Вона містить адреси блоків керування пам’яттю, перелік блоків керування пристроями MS-DOS, таблицю файлів тощо.

Інформація з векторної таблиці зв’язків може бути корисною для створення програм відображення розподілення пам’яті, виведення переліку завантажених драйверів, виведення переліку пристроїв прямого доступу до пам’яті та інших цілей.

У даній лабораторній роботі розглядатимемо роботу з векторною таблицею зв’язків ОС MS-DOS версії 6.22.

1 ПОСТАНОВКА ЗАДАЧІ

У даній лабораторній роботі потрібно виконати аналіз векторної таблиці зв’язків MS-DOS 6.22, у тому числі вивчити формат блоків керування пам’яті та навчитися працювати з ними (створювати, змінювати та видаляти).

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) виводити на екран адресу векторної таблиці зв’язків ОС MS-DOS 6.22 та значення її полів у зручному форматі;

2) виводити на екран перелік усіх блоків керування пам’яті зі вказуванням їхніх типів, розмірів та власників; у випадку великої кількості блоків повинно бути передбачено можливість посторінкового виведення;

3) виконувати запити на виділення пам’яті; у програмі повинно бути передбачено можливість уведення з боку користувача обсягу пам’яті, який потрібно виділити;

4) виконувати запити на зміну розміру арени пам’яті; у програмі повинно бути передбачено можливість уведення з боку користувача обсягу пам’яті, який займатиме арена пам’яті після зміни розміру;

5) виконувати запити на вивільнення виділеної арени пам’яті;

6) у будь-який момент роботи програма повинна мати можливість за запитом користувача виводити на екран перелік усіх блоків керу-

вання пам’яті;

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 Формат векторної таблиці зв’язків

Почнімо розгляд особливостей роботи з векторною таблицею зв’язків MS-DOS з огляду її формату.

Список полів векторної таблиці зв’язків наведено в таблиці 2.1.

Таблиця 2.1 – Формат векторної таблиці зв’язків

Зміщення, Б	Розмір, Б	Назва поля	Опис
–2	2	`mcb_seg`	Сегмент першого блоку керування пам’яттю (Memory Control Block, MCB)
0	4	`dev_cb`	Указівник на перший блок керування дисковими пристроями MS-DOS
4	4	`file_tab`	Указівник на таблицю файлів MS-DOS
8	4	`clock_dr`	Указівник на драйвер `CLOCK$`, установлений у файлі CONFIG.SYS або резидентний

Продовження таблиці 2.1

Зміщення, Б	Розмір, Б	Назва поля	Опис
12	4	`con_dr`	Указівник на драйвер CON, установлений у файлі CONFIG.SYS або резидентний
16	2	`max_btbl`	Максимальний розмір блоку (у байтах) для пристрою, що виконує передачу даних окремими блоками
18	4	`disc_buf`	Указівник на структуру, що описує дискові буфери
22	4	`drv_info`	Указівник на масив інформації про пристрої
26	4	`fcb_table`	Указівник на таблицю блоків керування файлами (File Control Block, FCB)
30	2	`fcb_size`	Розмір таблиці блоків керування файлами

Продовження таблиці 2.1

Зміщення, Б	Розмір, Б	Назва поля	Опис
32	1	`num_bdev`	Кількість реально використовуваних пристроїв, що виконують передачу даних окремими блоками
33	1	`lastdrive`	Значення `LASTDRIVE` у файлі CONFIG.SYS (за замовчуванням дорівнює 5)
34	?	`null_dr`	Початок драйвера `NUL` — першого в списку драйверів MS-DOS

Прокоментуймо призначення деяких полів:

- поле `mcb_seg` містить сегментну компоненту адреси першого *блоку керування пам'яттю* (Memory Control Block, MCB). Оскільки MCB завжди розташовують на границі параграфу (параграф дорівнює 16 байтам), повна адреса першого MCB дорівнює `mcb_seg`:0000. Знаючи адресу першого MCB, можна прослідкувати та змінити структуру блоків пам'яті (розділ 2.3). Наприклад, деякі віруси штучно зменшують розмір останнього вільного блоку пам'яті, записуючи у вивільнену ділянку своє тіло. Також, знаючи структуру MCB, можна видаляти з пам'яті непотрібні резидентні програми;

- поле `dev_cb` містить указівник на список блоків керування ди-

сковими пристроями MS-DOS. Кожен такий блок містить опис характеристик пристрою, а також указівник на відповідний драйвер. Блок керування пристроєм можна використовувати для низькорівневого доступу до диску чи для отримання довідкової інформації про пристрій;

- поле `file_tab` містить адресу таблиці файлів MS-DOS, яка для кожного відкритого файлу зберігає інформацію про кількість призначених файлу ідентифікаторів, режим використання файлу, номер першого кластера файлу тощо;

- поля `clock_dr` та `con_dr` дозволяє отримати доступ до драйверів `CLOCK$` та `CON`, відповідно. Це може знадобитися для організації виклику драйвера безпосередньо з програми;

- поле `drv_info` містить указівник на масив, який зберігає інформацію про дискові пристрої. Аналізуючи цей масив, програма може визначити кількість установлених у системі дискових пристроїв та їхні параметри;

- поле `fcb_table` містить указівник на таблицю *блоків керування файлами* (File Control Block, FCB). Розмір цієї таблиці записано в полі `fcb_size`. Його визначає оператор `FCBS` у файлі CONFIG.SYS;

- поле `lastdrive` містить значення параметра оператора `LASTDRIVE` у файлі CONFIG.SYS, або ж значення за замовчуванням. Уміст цього поля можна використовувати для визначення максимальної кількості дискових пристроїв у системі;

- поле `null_dr` містить адресу першого драйвера MS-DOS. Маючи цю адресу, можна прослідкувати весь список драйверів.

2.2 Адреса векторної таблиці зв'язків

Для визначення адреси, за якою векторну таблицю зв'язків розміщено в пам'яті, можна використати недокументовану функцію `52h` переривання `INT 21h`. Наприклад, мовою асемблера виклик цієї функції виглядає таким чином:

```
mov ax, 5200h
int 21h
```

За результатом виклику цієї функції регістри `ES:BX` міститимуть адресу поля `dev_cb` (таблиця 2.1). Для того, щоб отримати адресу поля `mcb_seg`, потрібно зменшити значення отриманого зміщення на 2 байти.

Приклад 2.1.

Розгляньмо вихідний текст програми на мові програмування C, що виводить на екран адресу векторної таблиці зв'язків (рисунок 2.1).

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

//структура для векторної таблиці зв'язків
typedef struct
{
    unsigned mcb_seg;
    void far *dev_cb;
    void far *file_tab;
    void far *clock_dr;
    void far *con_dr;
    unsigned max_btbl;
    void far *disk_buf;
    void far *drv_info;
    void far *fcb_tabl;
    unsigned fcb_size;
    unsigned char num_bdev;
    unsigned char lastdriv;
} CVT;

//спеціальний тип указівника для векторної таблиці зв'язків
```

```
typedef CVT  far* LPCVT;

void main(void)
{
    union REGS regs;
    struct SREGS sregs;
    LPCVT lpCVT;

    //очищуємо екран
    clrscr();

    //отримуємо адресу векторної таблиці зв'язків
    regs.h.ah = 0x52;
    intdosx(&regs, &regs, &sregs);

    //пересуваємо вказівник на поле msb_seg
    lpCVT = (LPCVT) MK_FP(sregs.es, regs.x.bx - 2);

    //виводимо на екран інформацію
    printf("\nCVT address: %Fp", (LPCVT) lpCVT);

    printf("\n\nPress any key...");
    getch();
}
```

Рисунок 2.1 – Результат виконання програми з Прикладу 2.1

У вищенаведеній програмі використано функцію `intdosx`, яка викликає переривання `21h`. У якості аргументів функція приймає адреси об'єднань `inregs` та `outregs`, а також адресу структури `segregs`. Для виклику функції спочатку потрібно записати в поля `inregs` значення потрібних регістрів. У результаті виконання функція запише в поля `outregs`

та `segregs` значення регістрів загального призначення та сегментних регістрів, відповідно, отримані після повернення з опрацьовувача переривання `21h`.

Також використано макрокоманду `MK_FP`, описану у файлі DOS.H. Вона дозволяє сформувати дальній указівник з наведених значень сегменту та зміщення.

2.3 Блоки керування пам'яттю в MS-DOS

2.3.1 Розподіл пам'яті в MS-DOS

У таблиці 2.2 представлено розподіл пам'яті в MS-DOS.

Таблиця 2.2 – Розподіл пам'яті в MS-DOS

Діапазон адрес	Уміст
0000:0000	Вектори переривань
0000:0400	Область даних BIOS
0000:0500	Область даних MS-DOS
xxxx:0000	Область програм MS-DOS, яка містить розширення BIOS, опрацьовувачі переривань MS-DOS, буфери, унутрішні структури даних MS-DOS, завантажувані драйвери пристроїв
xxxx:0000	Резидентна порція командного процесора COMMAND.COM
xxxx:0000	Резидентні програми
xxxx:0000	Запущені аплікації типу COM чи EXE

Продовження таблиці 2.2

Діапазон адрес	Уміст
xxxx:0000	Транзитна порція командного процесора COMMAND.COM
A000:0000	Пам’ять EGA, використовувана в деяких відеорежимах
B000:0000	Пам’ять монохромного відеоконтролера
B800:0000	Пам’ять відеоконтролера CGA
C800:0000	Зовнішній ПЗП
F600:0000	ПЗП інтерпретатора BASIC
FE00:0000	ПЗП BIOS

Прокоментуймо цю таблицю:

- перший кілобайт оперативної пам’яті займає таблиця векторів переривань, яка містить 256 елементів розміром по 4 байти — дальні адреси опрацьовувачів переривань;

- адреси 0000:0400–0000:04FF (або 0040:0000–0050:0000) займає область даних BIOS, яка містить унутрішні змінні BIOS. До них можна звертатися для отримання додаткової інформації. Формат цієї області різний для різних версій BIOS;

- починаючи з адреси 0000:0500 (або 0050:0000), розташовано область даних MS-DOS, у якій містяться внутрішні таблиці, змінні та структури даних ОС. Формат та розмір цієї області залежить від версії ОС;

- за областю даних MS-DOS розташовано велику область пам’яті MS-DOS, яка містить систему уведення-виведення MS-DOS (уміст файл IO.SYS), опрацьовувачі переривань, унутрішні буфери та структури даних ОС, завантажувані драйвери (зазначені у файлі CONFIG.SYS);

- після драйверів послідовно розташовано резидентну порцію командного процесора COMMAND.COM, резидентні програми, виконувану в даний момент аплікацію. Остання може займати решту пам’яті до адреси A000:0000 або тільки деяку її частину;

- нижню частину адресового простору (до адреси A000:0000) займає транзитна частина COMMAND.COM, яку може перекривати виконувана аплікація. Якщо програма перекриє частину COMMAND.COM, то по завершенню виконання програми цю частину командного процесора буде завантажено наново;

- область адрес A000:0000–C800:0000 використовують відеоконтролери;

- решту пам’яті до кінця першого МБ оперативної пам’яті займає область ПЗП. Також у цій області можуть міститися порти уведення/виведення деяких пристроїв, звернення до яких виконують аналогічно зверненню до пам’яті (так звані пристрої, відображені на пам’ять);

- адреси понад 1 МБ становлять розширену пам’ять (extended memory), яку MS-DOS використовує для організації електронного диску, кеш-пам’яті для дисків, завантаження резидентних програм та драйверів. У цій області пам’яті можуть зберігати свої дані деякі аплікації.

Зону пам’яті від області програм MS-DOS до відеопам’яті розбито на фрагменти, які називають *арени пам’яті* (memory arenas). Розмір арен обчислюється параграфами, тобто їхній розмір у байтах кратний 16. Арени пам’яті розташовано послідовно одну за одною. Кожна арена пам’яті на своєму початку (у нижніх адресах) має один параграф, відведений під блок керування пам’яттю (MCB). У наступному розділі розглянемо формат такого блоку.

2.3.2 Формат блоку керування пам’яттю

Блок керування пам’яттю має формат, представлений у таблиці 2.3.

Таблиця 2.3 – Формат блоку керування пам’яттю

Зміщення, Б	Розмір, Б	Назва поля	Опис
0	1	`type`	Тип блоку (M або Z)
1	2	`owner`	Сегментна компонента адреси власника блоку (оскільки власника завжди розташовано на границі параграфа, зміщення в цьому випадку завжди дорівнює 0) або 0, якщо блок описує сам себе, тобто відповідає вільній ділянці пам’яті
3	2	`size`	Кількість параграфів в арені пам’яті, яку описує даний блок
5	11	`reserved`	Зарезервовано

Блоки керування пам’яттю бувають двох типів:

- M (код 4Dh): проміжні блоки;
- Z (код 5Ah): останній блок (він може бути тільки один).

Кожен блок належить певній програмі. На самому початку роботи ОС має місце тільки один блок, який належить MS-DOS і є вільний. По мірі запиту в MS-DOS пам’яті з боку аплікацій, початковий блок розбивають на багато менших.

Кожен із блоків керування пам’яті містить сегментну компоненту адреси свого власника в полі `owner`. Це значення також називають *ідентифікатор процесу* (process ID). Із технічного погляду, за адресою `owner`:0000 розташовано *префікс сегменту програми* (Program Segment Prefix, PSP). Але, оскільки PSP безпосередньо передує програмному коду, і PSP разом із кодом

містяться в одній арені пам'яті, можна стверджувати, що початок PSP — це і є початок програми.

Якщо MCB у якості поля `owner` має значення 0, то відповідна арена пам'яті вільна.

Знаючи адресу першого MCB, можна отримати доступ до всіх інших блоків. Для цього потрібно зчитати з MCB розмір відповідної арени пам'яті та додати до цього числа 1. Наступний MCB знаходитиметься за сегментною адресою, утвореною як сума сегментної адреси попереднього MCB та отриманого щойно числа. Для блоку типу Z адреса, утворена вказаним чином, завжди вказує на кінець оперативної пам'яті.

Приклад 2.2.

Розгляньмо вихідний текст програми на мові програмування C, що виводить на екран усі блоки керування пам'яттю. Для кожного блоку програма виводить адресу блоку, його тип, адресу власника та розмір арени пам'яті (рисунок 2.2).

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

//структура для векторної таблиці зв'язків
typedef struct
{
    unsigned mcb_seg;
    void far *dev_cb;
    void far *file_tab;
    void far *clock_dr;
    void far *con_dr;
    unsigned max_btbl;
    void far *disk_buf;
    void far *drv_info;
    void far *fcb_tabl;
    unsigned fcb_size;
    unsigned char num_bdev;
    unsigned char lastdriv;
} CVT;

//спеціальний тип указівника для векторної таблиці зв'язків
typedef CVT far* LPCVT ;

//структура для блоку керування пам'яттю
typedef struct
{
    unsigned char type;
    unsigned owner;
```

```
    unsigned size;
    char reserve[11];
} MCB;

//спеціальний тип указівника для блоку керування пам'яттю
typedef MCB far* LPMCB;

LPMCB get_nmcb(LPMCB);

void main(void)
{
    union REGS regs;
    struct SREGS sregs;
    LPCVT lpCVT;
    LPMCB lpMCB;

    //очищуємо екран
    clrscr();

    //отримуємо адресу векторної таблиці зв'язків
    regs.h.ah = 0x52;
    intdosx(&regs, &regs, &sregs);

    //пересуваємо вказівник на поле msb_seg
    lpCVT = (LPCVT) MK_FP(sregs.es, regs.x.bx - 2);

    //отримуємо вказівник на перший блок керування пам'яттю
    lpMCB = (LPMCB) MK_FP(lpCVT->mcb_seg, 0);

    //виводимо на екран інформацію про блоки керування пам'яттю
    printf("\nMemory Control Blocks"
        "\nMCB Address Type Owner Size"
        "\n----------- ---- ----- ----"
        "\n");

    int counter = 0;
    for(;;)
    {
        counter++;

        if(counter % 10 == 0)
        {
            //для організації посторінкового виведення
            printf("Press any key...");
            getch();
            printf("\n");
        }

        if(lpMCB == NULL)
        {
            //якщо наштовхнулися на останній блок керування пам'яттю
            break;
        }

        printf("%Fp     %c        %04X    %04X\n",
            lpMCB, lpMCB->type,
            lpMCB->owner, lpMCB->size);
```

```
        //отримуємо адресу наступного блоку керування пам'яттю
        lpMCB = get_nmcb(lpMCB);
    }

    printf("\n\nPress any key...");
    getch();
}

//функція для отримання адреси наступного блоку керування пам'яттю
LPMCB get_nmcb(LPMCB mcb)
{
    unsigned seg, off;

    //перевіряємо тип блоку
    if(mcb->type == 'M')
    {
        //якщо це внутрішній блок

        //обчислюємо наступну адресу
        seg = FP_SEG(mcb) + mcb->size + 1;
        off = FP_OFF(mcb);

        return (MCB far*) MK_FP(seg, off);
    }
    else
    {
        //якщо це кінцевий блок
        return (LPMCB) NULL;
    }
}
```

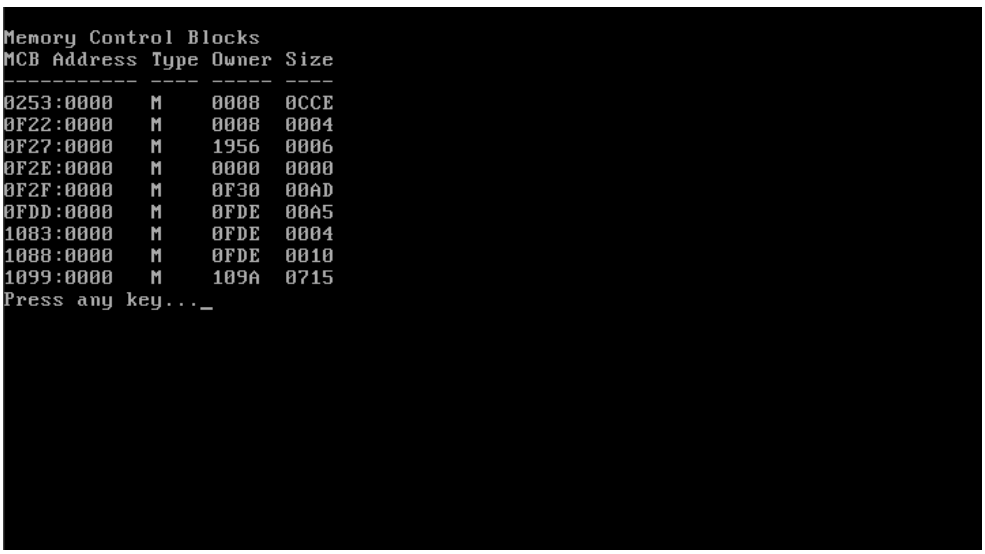

Рисунок 2.2 – Результат виконання програми з Прикладу 2.2

2.3.3 Динамічне керування пам’яттю

Практично всі програми використовують динамічне виділення пам’яті по мірі потреби, тому важливо знати, чи достатньо пам’яті в системі для запуску тієї чи тієї аплікації. Окрім того, оскільки MS-DOS виділяє пам’ять фрагментами, можлива фрагментація вільного простору: велика (сумарна) кількість пам’яті буде непотрібною, оскільки її буде розбито на багато ділянок невеликого розміру.

Для динамічного керування пам’яттю можна використовувати такі функції DOS (переривання `INT 21h`):

- `48h`: виділити арену пам’яті;
- `49h`: вивільнити арену пам’яті;
- `4Ah`: змінити розмір виділеної арени пам’яті.

За допомогою функції `48h` програма може запитати в DOS до 1 МБ пам’яті, хоча практично доцільно отримувати пам’ять сегментами по 64 КБ. З погляду структури програми виділена арена являтиме собою додатковий сегмент даних. Формат цієї функції такий:

- на вході:

  `BX` — розмір арени пам’яті (у параграфах);

- на виході:

  прапорець `CF` — установлено в 1, якщо під час виділення пам’яті сталася помилка, та в 0, якщо помилки не було;

  `AX` — у випадку успішного виконання функції містить сегментну адресу виділеної арени, у випадку помилки — код помилки;

  `BX` — у випадку помилки містить кількість вільних параграфів.

Наприклад, виділити програмі арену пам’яті розміром 3 параграфи мовою асемблера можна наступним чином:

```
mov ax, 4800h
mov bx, 3
int 21h
```

Тоді у випадку успішного виконання функції в регістрі `AX` матимемо се-

гментну адресу виділеної арени.

Особливості роботи функції 48h можна використати для визначення обсягу найбільшої вільної арени пам'яті. Для цього потрібно перед виконанням функції записати в регістр BX значення FFFFh. Виділення пам'яті очевидно завершиться з помилкою, і з регістра BX можна буде зчитати шукане значення.

Функцію 49h можна використовувати для вивільнення арени пам'яті, раніше виділеної за допомогою функції 48h. Неможливо вивільнити пам'ять, якою програма не володіє, або частину виділеної пам'яті (для цього потрібно використовувати функцію 4Ah). Формат функції 49h такий:

- на вході:

  ES — сегментна адреса арени, яку потрібно вивільнити;

- на виході:

  прапорець CF — установлено в 1, якщо під час виділення пам'яті сталася помилка, та в 0, якщо помилки не було;

  AX — у випадку помилки — її код.

Для зміни розміру арени пам'яті, виділеної раніше за допомогою функції 48h, можна використовувати функцію 4Ah. Формат цієї функції такий:

- на вході:

  BX — новий розмір арени пам'яті (у параграфах); може бути як більше, так і менше поточного;

  ES — сегментна адреса арени, розмір якої потрібно змінити;

- на виході:

  прапорець CF — установлено в 1, якщо під час виділення пам'яті сталася помилка, та в 0, якщо помилки не було;

  AX — у випадку помилки — її код;

  BX — у випадку помилки містить кількість вільних параграфів.

Функцію 4Ah зазвичай використовують для зменшення розміру програми до реально потрібного, оскільки DOS під час завантаження виділяє програмі всю наявну пам'ять.

Під час виконання вищевказаних функцій можуть мати місце такі помилки:

- код 7: блок керування пам'яттю знищено;

- код 8: недостатньо пам’яті для виконання функції;
- код 9: некоректна адреса арени пам’яті.

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 18] — М. : Диалог-МИФИ, 1995. — 254 с.

2. Финогенов К. Г. Самоучитель по системным функциям MS-DOS / К. Г. Финогенов. — [2-е изд., перераб. и дополн.] — М. : Радио и связь, Энтроп, 1995. — 382 с.

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

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