Звіт

# Виконання лабораторної роботи №8

Студента 2ПІ-25Б Заболотного Олександра Івановича

«Файли та робота з ними»

Тема: Файли та робота з ними

Мета: Опанувати введення/виведення даних у файли, робота з текстовими та бінарними файлами.

Задача 8. Створіть файл у форматі CSV input.csv, що містить відомості про абітурієнтів. Відомості про кожного абітурієнта — це прізвище, шифр спеціальності і оцінки, одержані на трьох вступних іспитах.   
При запуску програма запитує в користувача для кожної спеціальності прохідний бал і кількість місць.   
Програма видає відсортований список зарахованих абітурієнтів для кожної спеціальності відсортований за спаданням по середньому балу.   
Коли користувач увів всі необхідні данні, програма зберігає список зарахованих абітурієнтів у файл у форматі CSV result.csv, при цьому групує записи по спеціальностям і сортує за спаданням по середньому балу.

Текст програми:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_APPLICANTS 100

#define MAX\_SPECIALTIES 10

#define MAX\_NAME\_LEN 50

typedef struct {

    char surname[MAX\_NAME\_LEN];

    char specialty[10];

    int exam[3];

    float avg;

} Applicant;

typedef struct {

    char specialty[10];

    int passing\_score;

    int places;

} Specialty;

Applicant applicants[MAX\_APPLICANTS];

Specialty specialties[MAX\_SPECIALTIES];

int applicant\_count = 0, specialty\_count = 0;

void read\_applicants(const char \*filename) {

    FILE \*f = fopen(filename, "r");

    if (!f) { printf("File not found\n"); exit(1); }

    while (fscanf(f, "%49[^,],%9[^,],%d,%d,%d\n",

                  applicants[applicant\_count].surname,

                  applicants[applicant\_count].specialty,

                  &applicants[applicant\_count].exam[0],

                  &applicants[applicant\_count].exam[1],

                  &applicants[applicant\_count].exam[2]) == 5) {

        applicants[applicant\_count].avg =

            (applicants[applicant\_count].exam[0] +

             applicants[applicant\_count].exam[1] +

             applicants[applicant\_count].exam[2]) / 3.0f;

        applicant\_count++;

    }

    fclose(f);

}

void get\_specialty\_info() {

    specialty\_count = 0;

    for (int i = 0; i < applicant\_count; i++) {

        int found = 0;

        for (int j = 0; j < specialty\_count; j++) {

            if (strcmp(applicants[i].specialty, specialties[j].specialty) == 0) {

                found = 1;

                break;

            }

        }

        if (!found && specialty\_count < MAX\_SPECIALTIES) {

            strcpy(specialties[specialty\_count].specialty, applicants[i].specialty);

            specialty\_count++;

        }

    }

    for (int i = 0; i < specialty\_count; i++) {

        printf("Specialty code: %s\n", specialties[i].specialty);

        printf("Passing score: ");

        scanf("%d", &specialties[i].passing\_score);

        printf("Number of places: ");

        scanf("%d", &specialties[i].places);

    }

}

void process\_and\_save(const char \*filename) {

    FILE \*f = fopen(filename, "w");

    for (int s = 0; s < specialty\_count; s++) {

        Applicant eligible[MAX\_APPLICANTS];

        int count = 0;

        for (int i = 0; i < applicant\_count; i++) {

            if (strcmp(applicants[i].specialty, specialties[s].specialty) == 0 &&

                applicants[i].avg >= specialties[s].passing\_score) {

                eligible[count++] = applicants[i];

            }

        }

        for (int i = 0; i < count - 1; i++)

            for (int j = i + 1; j < count; j++)

                if (eligible[i].avg < eligible[j].avg) {

                    Applicant tmp = eligible[i]; eligible[i] = eligible[j]; eligible[j] = tmp;

                }

        fprintf(f, "Specialty: %s\n", specialties[s].specialty);

        for (int i = 0; i < count && i < specialties[s].places; i++)

            fprintf(f, "%s,%s,%.2f\n", eligible[i].surname, eligible[i].specialty, eligible[i].avg);

    }

    fclose(f);

}

int main() {

    read\_applicants("input.csv");

    get\_specialty\_info();

    process\_and\_save("result.csv");

    printf("Results saved to result.csv\n");

    return 0;

}

1. Призначення програми:

Програма призначена для автоматизованої обробки даних абітурієнтів, які зберігаються у файлі CSV. Вона дозволяє визначити прохідний бал і кількість місць для кожної спеціальності, формує та зберігає відсортований список зарахованих абітурієнтів для кожної спеціальності у новий CSV-файл.

2. Вхідні дані:

Вхідні дані — файл input.csv, що містить прізвище, шифр спеціальності та оцінки за трьома вступними іспитами для кожного абітурієнта. Додатково користувач вводить прохідний бал і кількість місць для кожної спеціальності.

3. Алгоритм роботи програми:

• Зчитування даних абітурієнтів з файлу CSV.

• Визначення унікальних спеціальностей серед абітурієнтів.

• Введення користувачем прохідного балу та кількості місць для кожної спеціальності.

• Відбір абітурієнтів, які набрали середній бал не менше прохідного.

• Сортування абітурієнтів за спаданням середнього балу для кожної спеціальності.

• Вибір найкращих абітурієнтів відповідно до кількості місць.

• Збереження результатів у файл result.csv з групуванням по спеціальностях.

4. Опис основних частин коду:

• #include <stdio.h> — для операцій введення/виведення.

• #include <stdlib.h> — для роботи з файлами та пам’яттю.

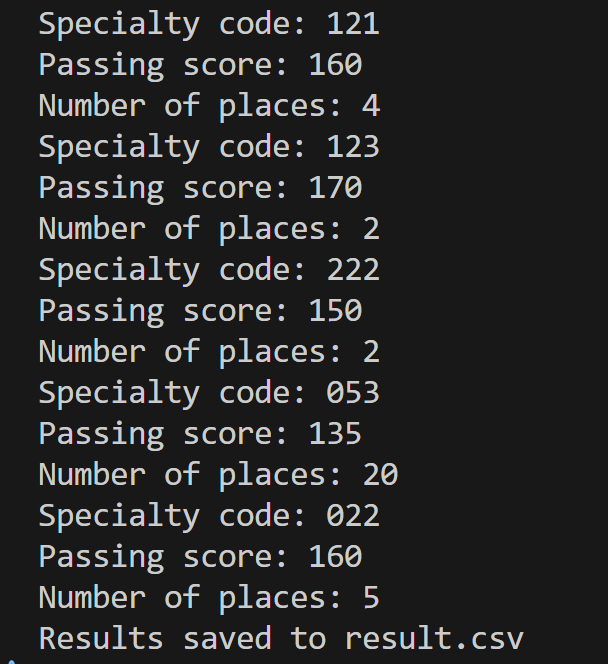
• #include <string.h> — для роботи з рядками.

• Оголошення структур для зберігання даних абітурієнтів та спеціальностей.

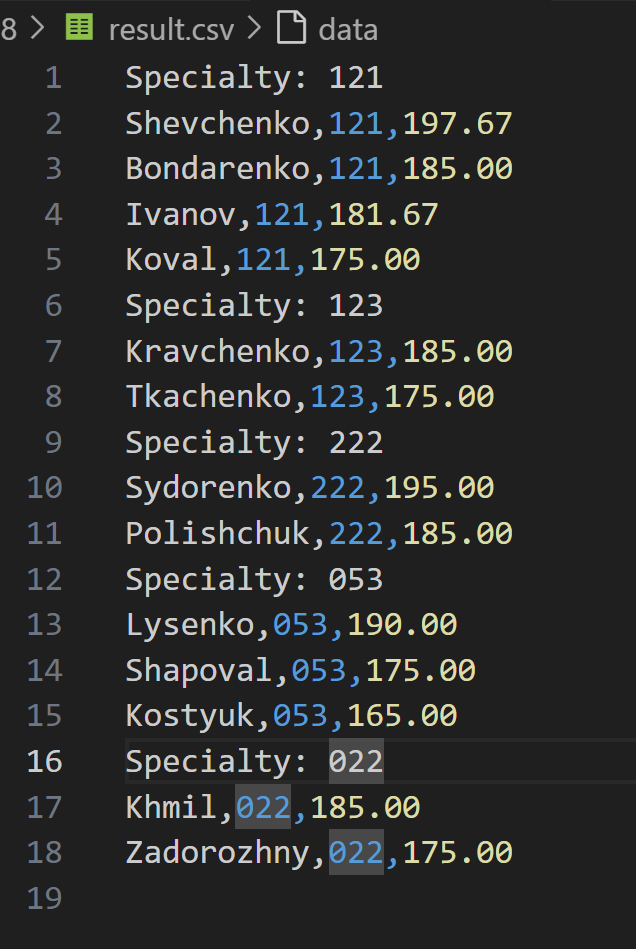
• Функції для зчитування даних з CSV, введення параметрів спеціальностей, обробки та збереження результатів.

• Головна функція для запуску всіх етапів програми.

Результат роботи:



Код з консолі



result.csv

Висновок: У ході виконання лабораторної роботи я навчився працювати з введенням та виведенням даних у файли у мові C, а також опанував роботу з текстовими файлами у форматі CSV. Я навчився зчитувати дані з файлу, обробляти їх у програмі, групувати та сортувати інформацію, а також зберігати результати у новий файл.

Отримані знання та навички стануть основою для подальшого вивчення роботи з файлами, зокрема з бінарними файлами, та для розробки більш складних програм, що потребують збереження і обробки великих обсягів даних.

Контрольні запитання:  
8. Поясніть реалізацію багато файлової програми за допомогою директиви #include.  
У багатофайлових програмах код розділяється на кілька файлів для зручності, модульності та повторного використання. Директива #include в мові C дозволяє включати вміст одного файлу в інший, що є ключовим для організації таких програм.

**Основні кроки реалізації:**

**Створення заголовкових файлів (.h):**

* + Заголовкові файли містять оголошення функцій, класів, констант, макросів тощо.
  + Вони включаються в інші файли за допомогою #include.
  + Для уникнення повторного включення використовуються протектори включення (include guards) або директива #pragma once.

9. Поясніть процес формування і виконання файлу проекта.

1. Написання вихідного коду

* Ти створюєш файл, наприклад main.c, з кодом на мові C.
* Це текстовий файл, який містить інструкції для компілятора.

2. Компіляція — перетворення в об'єктний код

Компілятор (наприклад, GCC або MSVC) виконує кілька внутрішніх фаз: Фаза препроцесора

* Обробляє директиви #include, #define, #ifdef тощо.
* Генерує **тимчасовий препроцесований файл**, наприклад main.i.
* Всі макроси розгортаються, а заголовки вставляються.

Фаза компіляції

* Преоброблений код перетворюється в **асемблерний код**, наприклад main.s.
* Це ще не машинний код, а інструкції для асемблера.

Фаза асемблювання

* Асемблер перетворює .s файл у **об'єктний файл** — main.o або main.obj.
* Це вже машинний код, але ще не готовий до запуску — без лінкування.

3. Лінкування — створення .exe

* Лінкер об'єднує всі .o файли (може бути кілька, якщо програма складається з модулів).
* Додає стандартні бібліотеки (наприклад, libc).
* Вирішує всі зовнішні посилання (наприклад, printf з stdio.h).
* Генерує **виконуваний файл** — main.exe.

10. Що таке логічний (фізичний) файл?  
**Логічний файл**— це те, як програміст або програма бачить файл:

* Це **послідовність байтів або записів**, яку читає або записує програма.
* У C це реалізується через стандартні функції: fopen(), fread(), fwrite(), fprintf(), fscanf() тощо.
* Наприклад, якщо ти відкриваєш файл data.txt і читаєш його рядок за рядком — це логічна структура.

Логічний файл може бути:

* **текстовим** — з символами, які можна прочитати (.txt, .csv)
* **бінарним** — з даними у форматі, який не призначений для читання людиною (.dat, .bin)

**Фізична організація** — це те, як файл зберігається на диску:

* Файл розбитий на **блоки пам’яті** (наприклад, 512 байт, 4 КБ).
* Операційна система розміщує ці блоки на фізичному носії (HDD, SSD).
* Файл може бути **фрагментованим** — тобто частини файлу можуть лежати в різних місцях диска.
* Файлова система (наприклад, NTFS, ext4) відповідає за зберігання, доступ і метадані (розмір, дата створення, права доступу).