**A blue and white logo

Description automatically generated**

**MINISTRY OF EDUCATION, CULTURE AND RESEARCH**

**OF THE REPUBLIC OF MOLDOVA**

**Technical University of Moldova**

**Faculty of Computers, Informatics and Microelectronics**

**Department of Software and Automation Engineering**

**Iamandii Ion student**

**Group: FAF-233**

**Report**

**Laboratory Work No.5**

***of the "Data Structures and Algorithms" course***

Checked:

teacher name, surname, academic level

Department of Software and Automation Engineering,

FCIM Faculty, UTM

Verificat:

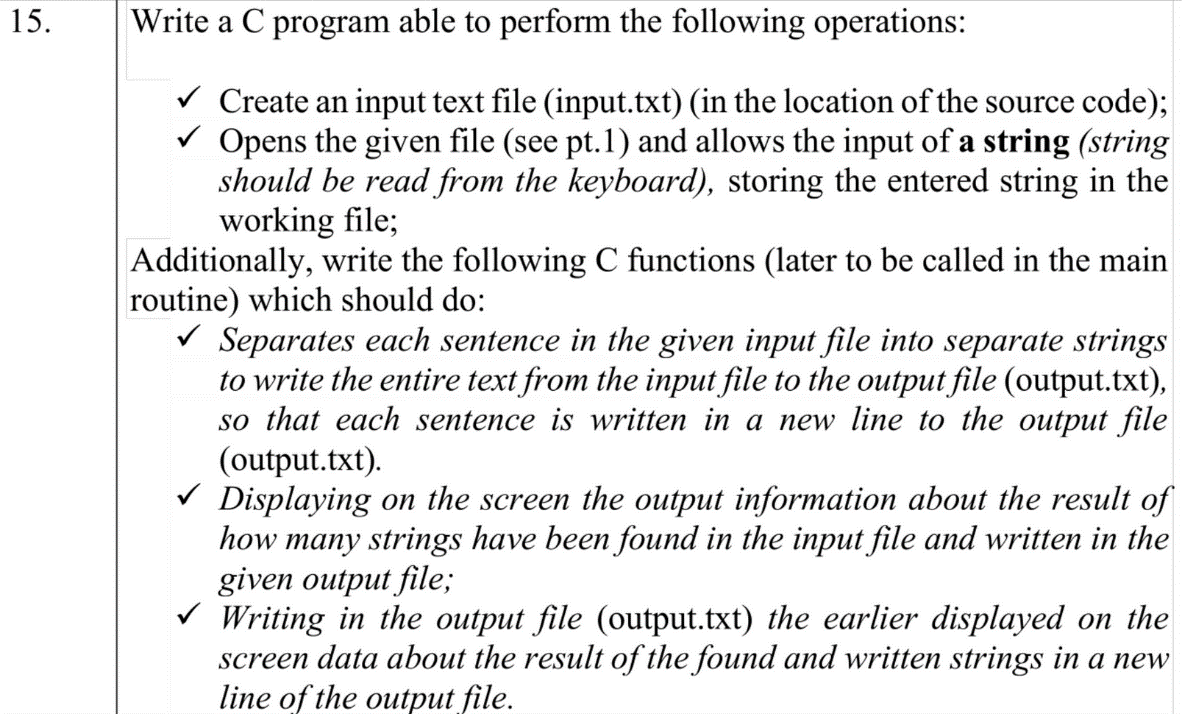
**Burlacu Natalia,** *PhD, associate professor*

Department of Software and Automation Engineering,

Facultatea FCIM, UTM

**Chisinau – 2024**

**Task 1**

****

**1. The code of the program, with relevant comments in it, and the Block diagram;**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

ssize\_t getline(char \*\*lineptr, size\_t \*n, FILE \*stream) {

    ssize\_t chars\_read = 0;

    ssize\_t bufsize = 0;

    int c;

    if (lineptr == NULL || n == NULL || stream == NULL) {

        return -1;

    }

    // Allocate memory if \*lineptr is NULL or bufsize is 0

    if (\*lineptr == NULL || \*n == 0) {

        \*n = 128;

        \*lineptr = (char \*)malloc(\*n);

        if (\*lineptr == NULL) {

            return -1; // Allocation failed

        }

    }

    // Read characters from stream until newline or EOF

    while ((c = fgetc(stream)) != EOF) {

        // Resize buffer if necessary

        if (chars\_read >= \*n - 1) {

            bufsize = \*n \* 2;

            char \*temp = (char \*)realloc(\*lineptr, bufsize);

            if (temp == NULL) {

                return -1; // Allocation failed

            }

            \*lineptr = temp;

            \*n = bufsize;

        }

        (\*lineptr)[chars\_read++] = (char)c;

        if (c == '\n') {

            break; // Stop reading at newline

        }

    }

    if (chars\_read == 0) {

        return -1; // No characters read

    }

    (\*lineptr)[chars\_read] = '\0'; // Null-terminate the string

    return chars\_read; // Return the number of characters read

}

// Function to check if a character is a sentence terminator

int isSentenceTerminator(char c) {

    return (c == '.' || c == '!' || c == '?');

}

// Function to separate sentences and write them to output file

void separateSentences(const char \*outputFileName, char \*input\_string, int length) {

    FILE \*outputFile = fopen(outputFileName, "w");

    char sentence[1024];

    int index = 0;

    int nrOfStrings = 0;

    for (int i = 0; i < length; i++) {

        char c = input\_string[i];

        if (c != '.' && c != '?' && c != '!') {

            sentence[index++] = c;

        } else {

            sentence[index] = '\0'; // Null-terminate the string

            fprintf(outputFile, "%s\n", sentence); // Write the sentence to the output file

            index = 0; // Reset the index for the next sentence

            nrOfStrings++;

        }

    }

    printf("Number of strings: %d", nrOfStrings);

    fprintf(outputFile, "%d\n", nrOfStrings);

    fclose(outputFile);

}

int main() {

    // Task 1

    // Create a file pointer

    FILE \*file\_ptr;

    // Open/create the file in write mode

    file\_ptr = fopen("C:/Users/MySurfacePro/Desktop/lab5/input.txt", "w");

    // Check if the file was opened successfully

    if (file\_ptr == NULL) {

        printf("Error opening file!\n");

        return 1; // Exit with error

    }

    // Task 2

    char \*input\_string = NULL;

    size\_t bufsize = 1024; // Initial buffer size

    input\_string = (char \*)malloc(bufsize \* sizeof(char));

    if (input\_string == NULL) {

        printf("Memory allocation failed!\n");

        return 1; // Exit with error

    }

    // Read string from keyboard

    printf("Input string\n");

    size\_t len = 0;

    int c;

    while ((c = getchar()) != EOF && c != '\n') {

        input\_string[len++] = c;

        if (len == bufsize) {

            bufsize \*= 2;

            input\_string = (char \*)realloc(input\_string, bufsize \* sizeof(char));

            if (input\_string == NULL) {

                printf("Memory reallocation failed!\n");

                return 1; // Exit with error

            }

        }

    }

    input\_string[len] = '\0'; // Add null terminator

    // Write the string to the file

    fputs(input\_string, file\_ptr);

    int length = strlen(input\_string);

    // Task 3 & 4

    file\_ptr = fopen("C:/Users/MySurfacePro/Desktop/lab5/output.txt", "w");

    separateSentences("C:/Users/MySurfacePro/Desktop/lab5/output.txt", input\_string, length);

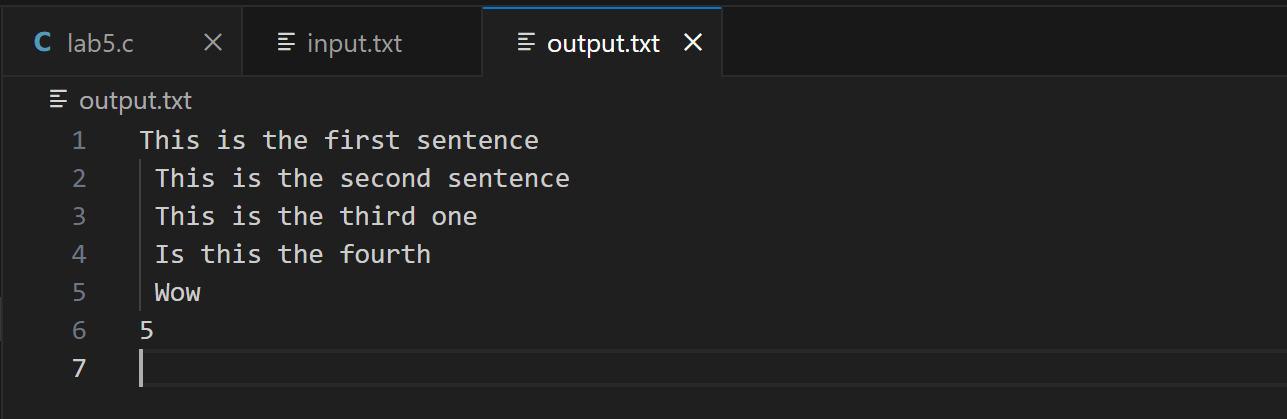
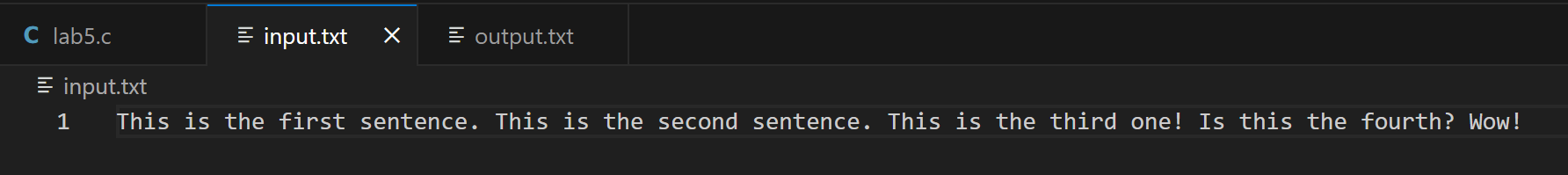
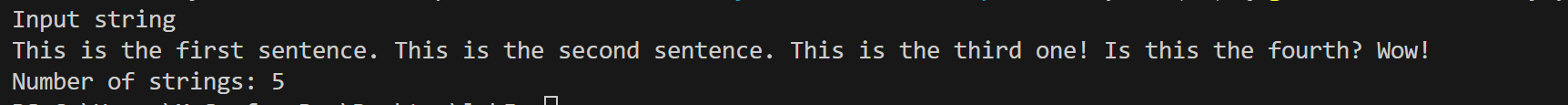
    free(input\_string);

    // Close the file

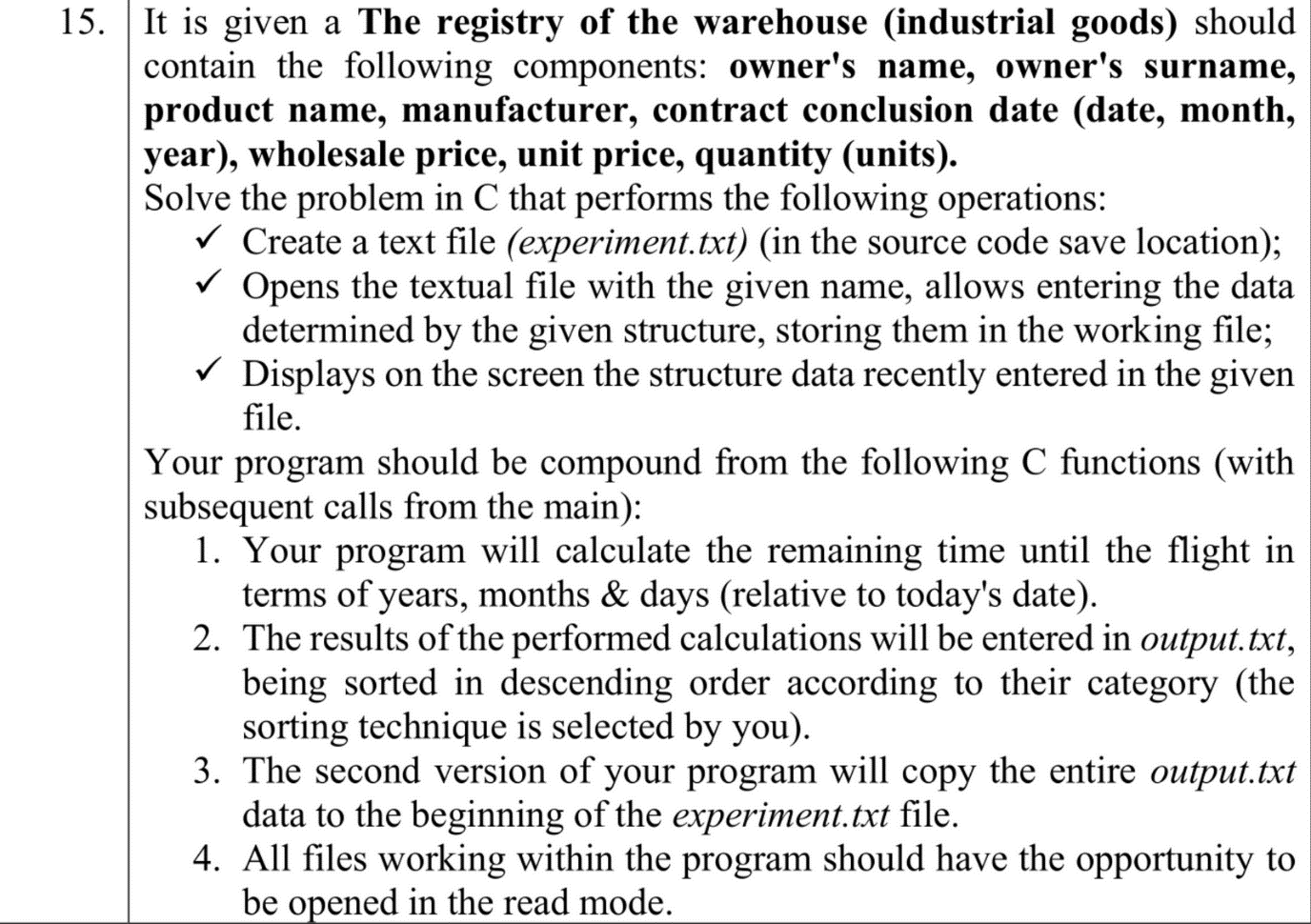
    fclose(file\_ptr);

    return 0;

}

**Ouput:**

**Task 2**

****

**1. The code of the program, with relevant comments in it, and the Block diagram;**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

struct WarehouseRecord {

    char owner\_name[50];

    char owner\_surname[50];

    char product\_name[50];

    char manufacturer[50];

    int contract\_date\_day;

    int contract\_date\_month;

    int contract\_date\_year;

    float wholesale\_price;

    float unit\_price;

    int quantity;

    int remaining\_years;

    int remaining\_months;

    int remaining\_days;

};

void calculateTimeUntilContractExpires(struct WarehouseRecord records[], int num\_records) {

    time\_t now;

    time(&now);

    struct tm \*current\_time = localtime(&now);

    for (int i = 0; i < num\_records; i++) {

        struct tm contract\_time = {0};

        contract\_time.tm\_year = records[i].contract\_date\_year - 1900;

        contract\_time.tm\_mon = records[i].contract\_date\_month - 1;

        contract\_time.tm\_mday = records[i].contract\_date\_day;

        time\_t contract\_timestamp = mktime(&contract\_time);

        double seconds\_diff = difftime(contract\_timestamp, now);

        records[i].remaining\_years = (int)(seconds\_diff / (365.25 \* 24 \* 3600));

        records[i].remaining\_months = (int)((seconds\_diff - records[i].remaining\_years \* 365.25 \* 24 \* 3600) / (30.44 \* 24 \* 3600));

        records[i].remaining\_days = (int)((seconds\_diff - records[i].remaining\_years \* 365.25 \* 24 \* 3600 - records[i].remaining\_months \* 30.44 \* 24 \* 3600) / (24 \* 3600));

    }

}

void sortRecords(struct WarehouseRecord records[], int num\_records) {

    // Sort records based on remaining time until contract expires

    for (int i = 0; i < num\_records - 1; i++) {

        for (int j = 0; j < num\_records - i - 1; j++) {

            // Calculate total remaining days

            int total\_remaining\_days1 = records[j].remaining\_years \* 365 + records[j].remaining\_months \* 30 + records[j].remaining\_days;

            int total\_remaining\_days2 = records[j + 1].remaining\_years \* 365 + records[j + 1].remaining\_months \* 30 + records[j + 1].remaining\_days;

            if (total\_remaining\_days1 < total\_remaining\_days2) {

                struct WarehouseRecord temp = records[j];

                records[j] = records[j + 1];

                records[j + 1] = temp;

            }

        }

    }

}

void printRecordsToFile(struct WarehouseRecord records[], int num\_records) {

    FILE \*output\_file = fopen("output.txt", "w");

    if (output\_file == NULL) {

        printf("Error opening file!\n");

        return;

    }

    // Define current\_time here

    time\_t now;

    time(&now);

    struct tm \*current\_time = localtime(&now);

    fprintf(output\_file, "Records sorted by remaining time until contract expires (Descending order):\n\n");

    for (int i = 0; i < num\_records; i++) {

        fprintf(output\_file, "Record %d:\n", i + 1);

        fprintf(output\_file, "Owner: %s %s\n", records[i].owner\_name, records[i].owner\_surname);

        fprintf(output\_file, "Product Name: %s\n", records[i].product\_name);

        fprintf(output\_file, "Manufacturer: %s\n", records[i].manufacturer);

        fprintf(output\_file, "Contract Date: %d-%02d-%02d\n", records[i].contract\_date\_year, records[i].contract\_date\_month, records[i].contract\_date\_day);

        fprintf(output\_file, "Wholesale Price: %.2f\n", records[i].wholesale\_price);

        fprintf(output\_file, "Unit Price: %.2f\n", records[i].unit\_price);

        fprintf(output\_file, "Quantity: %d\n", records[i].quantity);

        fprintf(output\_file, "Contract Expiration Date: %d-%02d-%02d\n",

                current\_time->tm\_year + 1900 + records[i].remaining\_years,

                current\_time->tm\_mon + 1 + records[i].remaining\_months,

                current\_time->tm\_mday + records[i].remaining\_days);

        fprintf(output\_file, "Remaining Time: %d years, %d months, %d days\n\n",

                records[i].remaining\_years, records[i].remaining\_months, records[i].remaining\_days);

    }

    fclose(output\_file);

}

void copyOutputToExperiment() {

    // Open output.txt for reading

    FILE \*output\_file = fopen("output.txt", "r");

    if (output\_file == NULL) {

        printf("Error opening output.txt!\n");

        return;

    }

    // Open experiment.txt for reading and writing

    FILE \*experiment\_file = fopen("experiment.txt", "r+");

    if (experiment\_file == NULL) {

        printf("Error opening experiment.txt!\n");

        fclose(output\_file);

        return;

    }

    // Open a temporary file for writing

    FILE \*temp\_file = fopen("temp.txt", "w");

    if (temp\_file == NULL) {

        printf("Error creating temporary file!\n");

        fclose(output\_file);

        fclose(experiment\_file);

        return;

    }

    // Copy contents of output.txt to temporary file

    int c;

    while ((c = fgetc(output\_file)) != EOF) {

        fputc(c, temp\_file);

    }

    // Append contents of experiment.txt to temporary file

    fseek(experiment\_file, 0, SEEK\_SET); // Move cursor to the beginning of experiment.txt

    while ((c = fgetc(experiment\_file)) != EOF) {

        fputc(c, temp\_file);

    }

    // Close all files

    fclose(output\_file);

    fclose(experiment\_file);

    fclose(temp\_file);

    // Replace experiment.txt with the contents of the temporary file

    remove("experiment.txt");

    rename("temp.txt", "experiment.txt");

    printf("Output copied to experiment.txt successfully.\n");

}

int main() {

    // Create and open experiment.txt for writing

    FILE \*experiment\_file = fopen("C:/Users/MySurfacePro/Desktop/lab5pt2/experiment.txt", "w");

    if (experiment\_file == NULL) {

        printf("Error opening file!\n");

        return 1;

    }

    int num\_records;

    printf("Enter the number of records to enter: ");

    scanf("%d", &num\_records);

    // Array to hold warehouse records

    struct WarehouseRecord records[num\_records];

    // Input warehouse registry data

    for (int i = 0; i < num\_records; i++) {

        printf("Record %d:\n", i+1);

        printf("Enter owner's name: ");

        scanf("%s", records[i].owner\_name);

        printf("Enter owner's surname: ");

        scanf("%s", records[i].owner\_surname);

        printf("Enter product name: ");

        scanf("%s", records[i].product\_name);

        printf("Enter manufacturer: ");

        scanf("%s", records[i].manufacturer);

        printf("Enter contract conclusion date (day month year): ");

        scanf("%d %d %d", &records[i].contract\_date\_day, &records[i].contract\_date\_month, &records[i].contract\_date\_year);

        printf("Enter wholesale price: ");

        scanf("%f", &records[i].wholesale\_price);

        printf("Enter unit price: ");

        scanf("%f", &records[i].unit\_price);

        printf("Enter quantity (units): ");

        scanf("%d", &records[i].quantity);

    }

    // Input warehouse registry data and write to experiment.txt

    for (int i = 0; i < num\_records; i++) {

        fprintf(experiment\_file, "Record %d:\n", i + 1);

        fprintf(experiment\_file, "Name: %s\nSurname: %s\nProduct Name: %s\nManufacturer: %s\nContract Date: %d-%d-%d\nWholesale Price: %.2f\nUnit Price: %.2f\nQuantity: %d\n",

                records[i].owner\_name, records[i].owner\_surname,

                records[i].product\_name, records[i].manufacturer,

                records[i].contract\_date\_day, records[i].contract\_date\_month, records[i].contract\_date\_year,

                records[i].wholesale\_price, records[i].unit\_price, records[i].quantity);

        fprintf(experiment\_file, "\n");

    }

    printf("----------------------------------------------------------------------------------------------------------------------------------\n");

    printf("| %-8s | %-12s | %-20s | %-15s | %-12s | %-10s | %-10s | %-8s |\n", "Record", "Owner", "Product Name", "Manufacturer", "Contract Date", "Wholesale", "Unit", "Quantity");

    printf("----------------------------------------------------------------------------------------------------------------------------------\n");

    for (int i = 0; i < num\_records; i++) {

        printf("| %-8d | %-12s %s | %-20s | %-15s | %02d-%02d-%04d | %-10.2f | %-10.2f | %-8d |\n",

            i + 1, records[i].owner\_name, records[i].owner\_surname,

            records[i].product\_name, records[i].manufacturer,

            records[i].contract\_date\_day, records[i].contract\_date\_month, records[i].contract\_date\_year,

            records[i].wholesale\_price, records[i].unit\_price, records[i].quantity);

    }

    printf("----------------------------------------------------------------------------------------------------------------------------------\n");

    // Open output.txt for writing

    FILE \*output\_file = fopen("output.txt", "w");

    if (output\_file == NULL) {

        printf("Error opening file!\n");

        return 1;

    }

    // Calculate remaining time until contract expires

    calculateTimeUntilContractExpires(records, num\_records);

    // Sort records based on remaining time

    sortRecords(records, num\_records);

    // Print sorted records to output.txt

    printRecordsToFile(records, num\_records);

    copyOutputToExperiment();

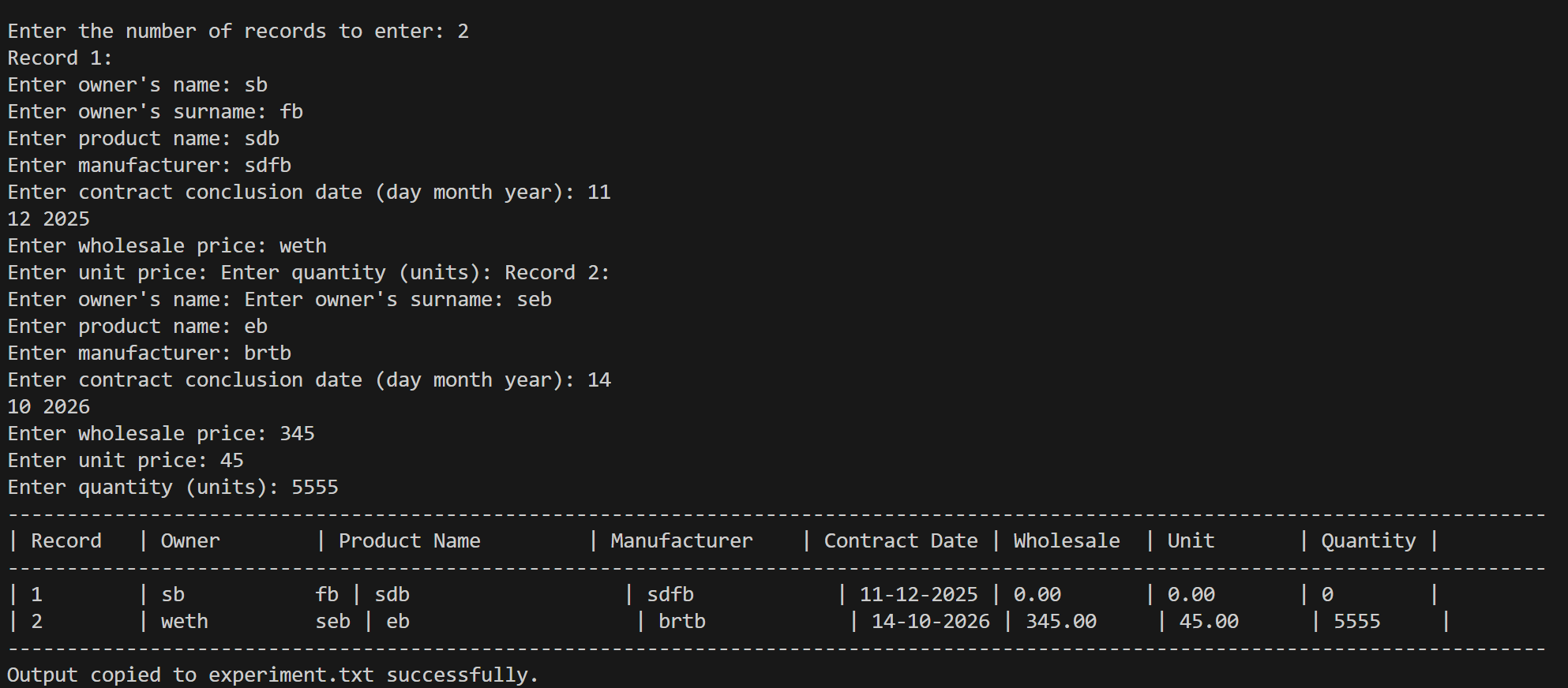
    fclose(output\_file);

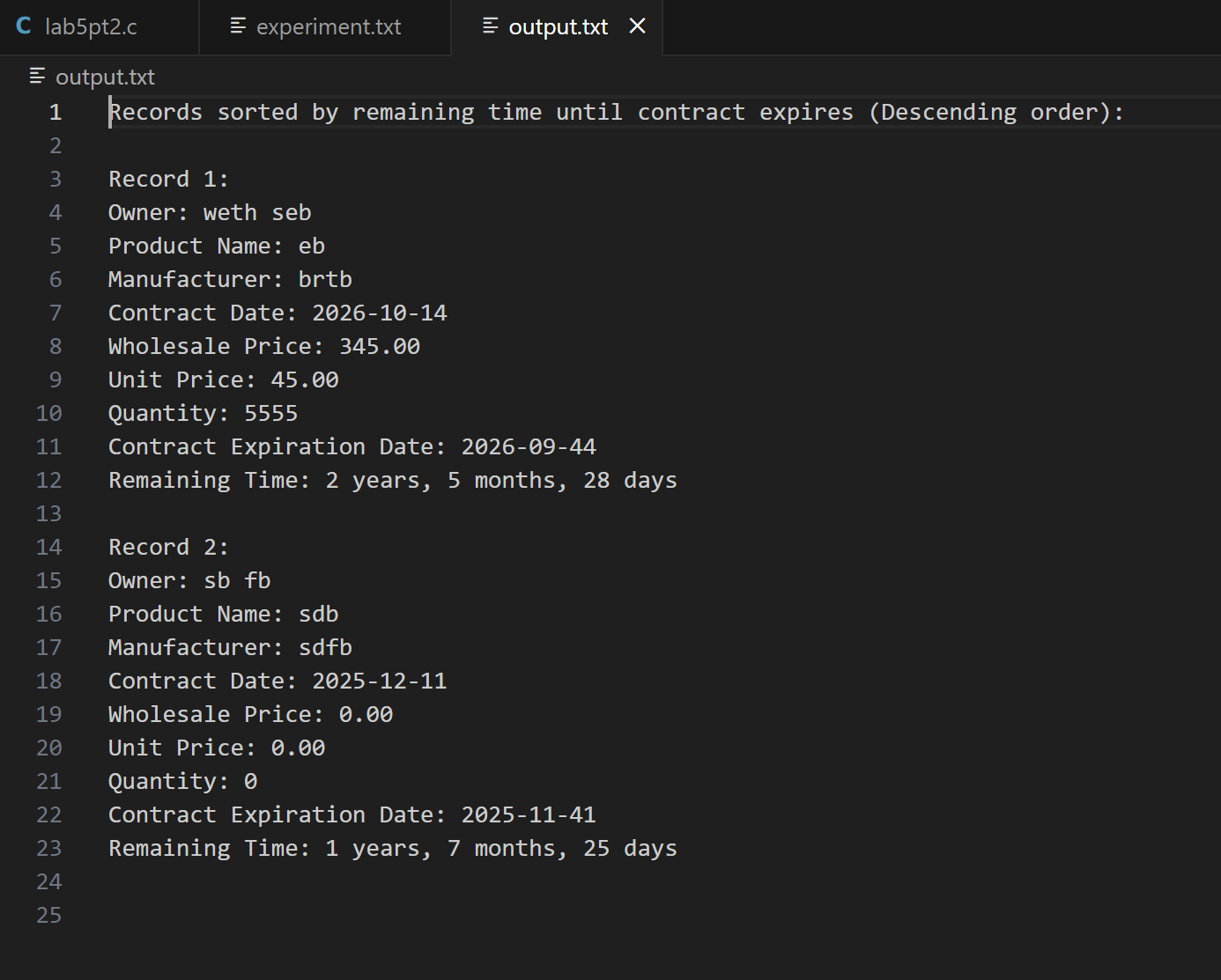
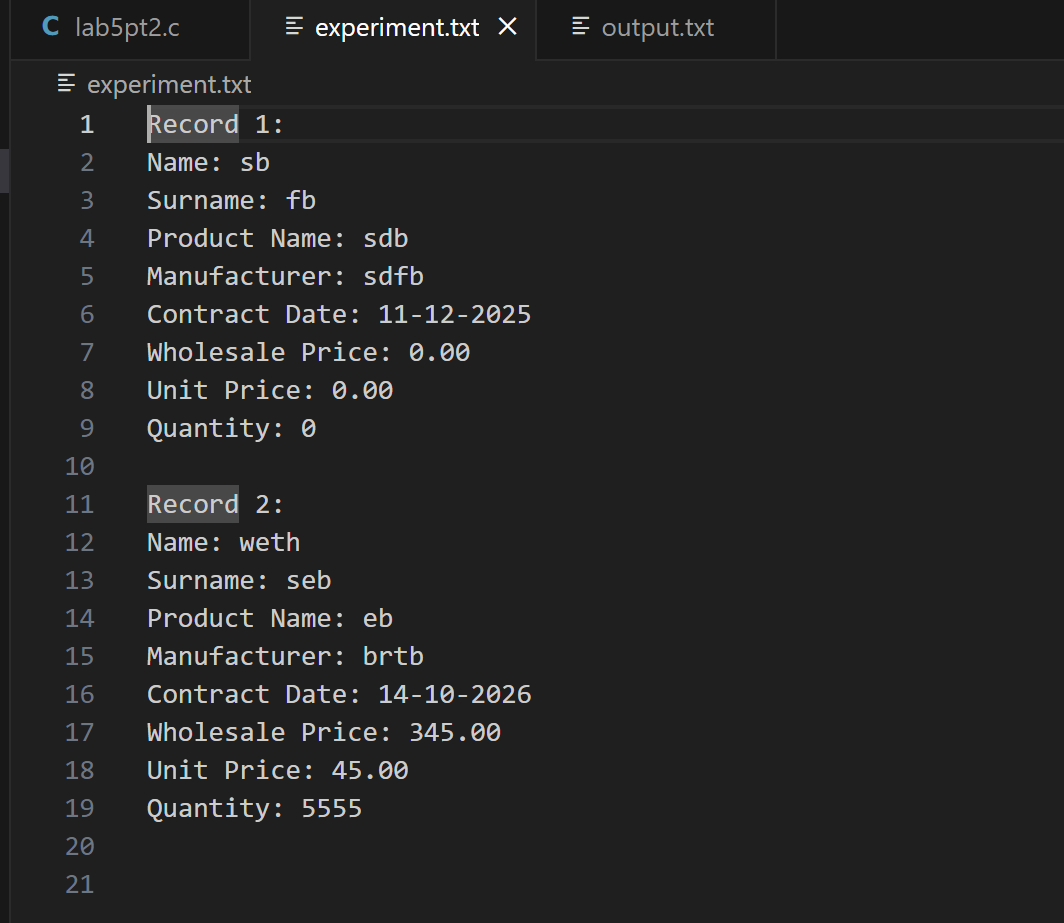
    fclose(experiment\_file);

    return 0;

}

**Output:**

****

****

**Conclusion:**

Combining file handling and structures in C facilitates efficient data management by organizing related information into structured units. Structures provide clarity and organization to data, while file handling operations enable reading from and writing to external files. Together, these concepts empower programmers to build robust applications for tasks like data processing and database management. By defining appropriate structures, complex data entities can be modeled, making it easier to work with them in code. The synergy between file handling and structures allows for the creation of efficient, organized, and scalable solutions for a wide range of programming challenges.