Lab Sheet No 2. Files

1) Objectives

The objective of this lab is to learn, through a set of activities and practical exercises, the various aspects of file handling in the C language. By the end of this lab, the student should be familiar with reading from and writing to files, handling file pointers, and managing file operations correctly in C.

2) Example 1: Reading and appending a file

The following program, once completed, allows reading integers from the file `"E:/numbers.txt"`, calculates their sum, and appends the sum at the end of the same file.

```
#include <stdio.h>
int main() {
    FILE *file;
    char filename[50]="E:/numbers.txt";
    int num, sum = 0;
    // Open the file in append mode
    file = fopen(filename, "....");
    if (file == NULL) {
        printf("Error opening the file %s.\n", filename);
    }
    else{
        // Read integers from file and calculate sum
        while (fscanf(file, "%d", &num) == 1) {
            ....;
        }
        // Move file pointer to end of file
        fseek(file, 0, ....);
        // Append the sum to the file
        fprintf(...., "....", ....);
        // Close the file
        ....(file);
    }
    return 0;
}
```

1. Create a new project and type the provided code snippet
2. Complete the code to achieve the intended functionality.

3. Modify the program so that the calculated sum is also saved in another file named "E:/Sum.txt".

3) Application Exercises

1. Given a text file "data.txt" containing a list of integers, write a program to find and display the maximum and minimum values in the file.

   Implement functions to read integers from the file, find the maximum and minimum values, and display them on the screen.

2. Write a program that copies the contents of one text file "source.txt" to another text file "destination.txt".

   Implement functions to read from "source.txt", write to "destination.txt", and handle any errors that may occur during file operations.

3. Write a program to merge two sorted text files "file1.txt" and "file2.txt" into a new sorted file "merged.txt".

   Implement functions to read from both files, merge their contents while maintaining sorted order, and write the sorted output into "merged.txt".

4. Consider an invoice saved as a structured file consisting of a sequence of lines, each representing a command. Each line contains: a customer name, an item name, and the unit price of the item. An example of such a file is as follows:

```
Achouri   PC       42000
Khemmal   Ecran    18000
Selatnia  Table    7000
```

   Write a program that reads the invoice command by command, asks the user to enter the number of items purchased for each command via keyboard input, calculates the command total, and saves all the information in another file. Each line in the resulting file should contain: the customer name, the item name, the unit price, the number of items purchased, and the command total.

   The resulting file should start with the header "Customer Item UP Nb Total", and end with the total amount of the invoice. An example of the resulting file is:

```
Customer   Item     UP        Nb   Total
-----------------------------------------------
Achouri    PC       42000.00  3    126000.00
Khemmal    Ecran    18000.00  2    36000.00
Selatnia   Table    7000.00   10   70000.00
-----------------------------------------------
Invoice total: 232000.00
```

4) Additional Exercises

1. Implement a program that reads a text file `"employees.txt"` containing employee details (ID, Name, Salary) separated by `';'` and calculates the total salary expenditure for all employees.

   Use file handling operations to parse the text file, calculate the total salary, and display it on the screen.

2. Write a program that reads a text file **`"words.txt"`** and counts the occurrences of each word in the file. Display the results in alphabetical order.

   Implement functions to read from the file, tokenize words, count their occurrences using data structures like arrays or linked lists, and display the results sorted alphabetically.

3. Create a program that reads a text file `"message.txt"`, encrypts its contents using a simple encryption technique (e.g., shifting characters by a fixed number), and writes the encrypted message to another file `"encrypted.txt"`.