Prerequisites:

- Basic knowledge of using computer and text editor
- Knowledge how to create a program
- Knowledge how to create a function
- Knowledge of different types
- Knowledge about conditional statements
- Knowledge about loops

Aims:

In this laboratory student will learn how to:

- save data into a file
- open data from a file

Tab. 1.1: Possible modes to open a file

Mode	Description
`r`	opening a file to read-only; system sets operation position's pointer at the beginning of the file
`r+`	as above, however, it is possible also writing data into a file
`w`	opening a file to write-only; if file does not exist it is created a new file, if such a file exists it is cleared to be empty
`w+`	as above, however, it is possible also reading from file (of course if any data were already added to such a file in the current session)
`a`	opening a file to write-only, however, content which was in this file before is not cleared and the operation position's pointer is set at the end of this file (that is why a new data are added to existing one); if such a file does not exist it is created
`a+`	as above, however, it is also possible reading some data from this file

1 Theoretical background

1.1 Opening a disk file

To open some file at first some pointer should be defined. Such a pointer allows to save (redirect) any stream into a file. In earlier laboratories you have already used some streams when `printf()` and `scanf()` functions were used. Example source code which allows to open some file is given bellow:

```
FILE *FileWithData;
...
FileWithData = fopen(name, type);
```

Name of the file (`name`) could be as follows:

- **only a file name** – for example `data.txt`
- **relative name** – for example `MySubfolder/data.txt`
- **absolute name** – for example `/home/student/Desktop/Lab07/MySubfolder/data.txt`

In turn, types of operations (`type`) are given in Tab. 1.1. Additionally, letter `b` could be added to determine a file opening mode. It is used when we want to use binary reading and writing to a file.

If you want to open an existing file `data.txt` in a read-only mode and to create a new

file `results.txt` in an appending mode you have to create two variables and you have to call twice `fopen()` with the following parameters:

```
FILE *FileWithData, *FileWithResults;
...
FileWithData = fopen("data.txt", "r");
FileWithResults = fopen("results.txt", "a");
```

It should be noted that user can has no privilege to open some file. In such a case pointer to struct `FILE` does not exist and it is empty. Because errors with I/O operations are very common therefore you should always check result given by `fopen()` pointer. Bellow is presented a relevant example:

```
FILE *FileWithData, *FileWithResults;
...
FileWithData = fopen("data.txt", "r");
if (!FileWithData)
{
  fprintf(stderr, "Error opening a file!\n");
  return 0;
}
FileWithResults = fopen("results.txt", "a");
if (!FileWithResults)
{
  fclose(FileWithData);
  fprintf(stderr, "Error creating a file!\n");
  return 0;
}
```

1.2 Closing a disk file

If an access to some disk file in no longer needed such a file should be closed as soon as possible. Closing file allows to save all changes which were done in such a file and it allows also to release some memory which was reserved by the standard C library used to operate with file's variable. Thanks that other programs can have right now access to this file.

To close a file `fclose()` subprogram should be called. The only argument of this function is the name of the file (`FILE` type) which will be closed. Typically, operations with file looks as follows:

```
FILE *FileWithData;
...
FileWithData = fopen("MyFile", "r");
if (FileWithData)
{
  ...
  fclose(FileWithData);
}
```

Sometimes in the next part of the source code it could be some reference to the `FileWithData` variable therefore it is wise to set it to `0`:

```
fclose(FileWithData);
FileWithData = 0;
```

1.3 Saving data into a file

To save some data into a disk file the `fprintf()` or the `fputs()` functions could be used, similarly as the `printf()` function was used to write something in the console. To write something into a disk file this file has to be open in `w` or `a` mode. In mode `w` a new text replace previous one stored in a file; in mode `a` a new text is added at the end of the file and the old content of the file is not deleted. The following source code allows to add some text to content included in some file:

```
#include <stdio.h>

int main()
{
  FILE *FileWithData;
  char Row[256];
  // Open some file and check if everything is all right.
  FileWithData = fopen("journal.txt", "a");
  if (!FileWithData)
  {
    fprintf(stderr, "Error opening journal.txt file!\n");
    return 1;
  }
```

```
    // Add new rows to file. At first add additionally one empty row
    // which allows to separate different parts.
    printf("\nProvide a new parts. Empty row ends provideng\n\n");
    fputs("\n", FileWithData);
    for (;;)
    {
      fgets(Row, sizeof(Row), stdin);
      if (Row[0] == '\n')
      {
        // Close file.
        printf("Adding a new data is finished.\n");
        fclose(FileWithData);
        return 0;
      }
      fputs(Row, FileWithData);
    }
    return 0;
}
```

1.4 Reading data from a file

To read some data from a file the `fgets()`, `fgetc()` or the `fscanf()` functions could be used, similarly as the `scanf()` function was used to read something from the console. If it is known how long is the file it is not problem, however, in case when we do not know length (number of lines) in file an additional `feof()` function should be used.

1.5 Reaching end of the file

To determine if end of the file was achieved the `feof()` function should be used. The only parameter is the file's variable. This subprogram returns logical vale equal to true if an end of the file was reached; or logical value equal to false if it is still possible to reading a data from this file. The commonly made mistake is as follows:

```
while (!foef(FileWithData))
{
  /* Read some portion of data. */
  ...
```

```
    /* Work/execute with this portion of data. */
    ...
  }
```

Before reading it is not possible to decide if the end of the file was reached. And if it was achieved it is not possible to secure in case when not fully portion of data will be executed. Reaching the end of the file should be checked just after try of reading this file, for example:

```
for (;;)
{
  /* Read some portion of data. */
  ...
  /* Is the end of the file? */
  if (feof(FileWithData)) break;
  /* Work/execute with this portion of data. */
  ...
}
```

2 Exercises

1. Write a source codes for examples explained above (there will be few programs).
2. Create a new file and read it in your program displaying its content in the terminal.
3. Generate randomly 50 values and display they in the terminal. Add possibility to store these values in the file `random.dat`.
4. Open file `random.dat` and sort values from the smallest to the biggest one and display the result of your calculation in the terminal.
5. Open file `random.dat` and sort values from the smallest to the biggest one and save result in this same file.
6. Create two new files and add randomly text to these files. Right now open both files in your program and merge them together (at first it will be content of the first file, then content of the second file). Results will be displayed in the terminal and also will be saved as the `result1.txt` file as well.
7. Repeat task no. 6, however, this time merge file in a way: first line of the first file then first line of the second file, next second line of the first file then second line of the second file, and so on. Save result as the `result2.txt` file.