Laboratory: USING FILES
I. THEORETICAL ASPECTS

1. Introduction

You are used to entering the data your program needs using the console but this is a time consuming task. Using the keyboard is difficult for entering large data sets and it also opens up the opportunity of user error. Furthermore you may want to run your program on large data batches automatically. For these reasons most programming languages offer you the opportunity to read the input data from a file.

In order to effectively use files you need to understand a few core concepts about them. First of all every file needs to have a **path** but there are two types of path. There is a **relative path** and an **absolute path**.

The **relative path** is defined from the folder where your program is running.
Examples:
"data1.txt" this text file is considered to be in the same folder as the running program
"hah\data2.txt" this text file is in the hah subfolder of the folder where the program is running
"..\pop\data3.txt" this text file is in the pop subfolder that exists one level up from the folder where the program is running

The **absolute path** is defined starting from the root of the drive you want to access. An absolute path always starts with the letter designation of the drive (most commonly C)
Examples:
"C:\student\program\data.txt" this text file is on the C drive in the student folder, program subfolder
"E:\data2.txt" this text file is on the E drive in the root. **Warning**: Placing files in the root of the drive is not recommended and may cause conflicts with the operating system.

If you try to access a file without issuing the correct path then the operation will fail. In some programming environments this will result in an error while in other the error is simply suppressed and no data is accessed.

When you are certain of the path of the file you must decide if you want to *open* the file for **reading** data or you are opening the file for **writing** data. Simultaneous read/write access is usually possible but it should be avoided in order to not cause conflicts.
Once the file has been opened it must be subsequently closed in order for other programs to access the data stored inside it. Computer operating systems restrict the access to a file one program at a time. Furthermore if your program writes data to a file and does not properly close it before terminating the execution there is a high likelihood that the data will not be saved and the changes will be lost.

Here we present a pseudocode to better understand the workflow with data files:

Reading:

Define a *File_variable;*
File_variable=File open (path, read);
File readfunction(*File_variable*, content to read);
File close (*File_variable*);

Writing:

Define a *File_variable;*
File_variable=File open (path, write);
File writefunction(*File_variable*, content to write);
File close (*File_variable*);

To define a file variable we must simply write FILE *fisier;
Notice that the file variable is a pointer!

The open file function is called fopen and one particular peculiarity is that the function returns a file to our file variable. Consequently we must write it as fisier=fopen("data.txt",r); to open the file "data.txt" located in the same folder as the running program for reading.

Here we have the complete list of file handling functions (all belong to **stdio.h**):

Function	**description**
FILE = fopen(PATH,MODE)	create a new file or open a existing file
fclose(FILE)	closes a file
fgetc(FILE)	reads a character from a file
fputc(FILE)	writes a character to a file
fscanf(FILE, CONTENT)	reads a set of data from a file
fprintf(FILE, CONTENT)	writes a set of data to a file
fgets(…)	reads a integer from a file
fputs(…)	writes a integer to a file
fseek(…)	set the position to desire point
ftell(…)	gives current position in the file
rewind(FILE)	set the position to the begining point

For each file we can open it in a different mode. Here we present the complete list of modes available to developers:

mode	description
r	opens a text file in reading mode
w	opens or create a text file in writing mode.
a	opens a text file in append mode
r+	opens a text file in both reading and writing mode
w+	opens a text file in both reading and writing mode
a+	opens a text file in both reading and writing mode
rb	opens a binary file in reading mode
wb	opens or create a binary file in writing mode
ab	opens a binary file in append mode
rb+	opens a binary file in both reading and writing mode
wb+	opens a binary file in both reading and writing mode
ab+	opens a binary file in both reading and writing mode

II. ASSIGNMENT WORKFLOW

First let us programmatically create a file:

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
   FILE * fp;

   fp = fopen ("data1.txt", "w");
//if the file does not exist it is created by the write mode

   fprintf(fp, "%s %s %s %d", "We", "are", "in", 2017);

   fclose(fp);

   return(0);
}
```

Now that we have created a file we should read it.
Create a new program (in the same folder) that reads the data from the file:

```
#include <stdio.h>
#include <stdlib.h>


int main()
{
   char str1[10], str2[10], str3[10];
   int year;
   FILE * fp;

   fp = fopen ("data1.txt", "r");

   fscanf(fp, "%s %s %s %d", str1, str2, str3, &year);

   printf("Read String1 |%s|\n", str1 );
   printf("Read String2 |%s|\n", str2 );
   printf("Read String3 |%s|\n", str3 );
   printf("Read Integer |%d|\n", year );

   fclose(fp);

   return(0);
}
```

Now please modify the file with notepad. Change the year for example or change the words. After saving the modified file in notepad run the reading program again.

Now run the writing program again.

What happened to the data in the file?

Play around with these two programs. One to read and one to write.

Can you integrate them into a single program that writes some data to a file then reads it?

What if we don’t know the format of the file we are reading? Create a new file using notepad containing some random text. Save it as file.txt.
In this case it would be a good idea to read the file character by character:

```
#include <stdio.h>

int main ()
{
   FILE *fp;
   int c;

   fp = fopen("file.txt","r");
   while(1)
   {
      c = fgetc(fp);
      if( feof(fp) )
      {
         break ;
      }
      printf("%c", c);
   }
   fclose(fp);

   return(0);
}
```

Notice the infinite loop that is exited using break when the End Of File (abbreviated to EOF) is detected. This strange construction is necessary because we do not know how big the file is.

What about writing character by character? The fputc function helps us with this:

```
#include <stdio.h>

int main ()
{
   FILE *fp;
   int ch;

   fp = fopen("file.txt", "w");
   for( ch = 33 ; ch <= 100; ch++ )
   {
      fputc(ch, fp);
   }
   fclose(fp);

   return(0);
}
```

Try reading the file you created with this program. What does it contain?

Working on a character by character basis is fine but what about whole strings of characters?

```
#include <stdio.h>

int main ()
{
   FILE *fp;

   fp = fopen("file2.txt", "w");

   fputs("We have opened this file.", fp);
   fputs("And we are writing to it", fp);
   fputs("Resistance is futile", fp);

   fclose(fp);

   return(0);
}
```

When you open this file in notepad are all the strings on the same line or are they on different lines?

What about reading a string of characters?

```
#include <stdio.h>

int main()
{
   FILE *fp;
   char str[60];

   fp = fopen("file2.txt" , "r");
   if(fp == NULL)
   {
      printf("Error opening file");
      return(-1);
   }
   fgets (str, 60, fp);
   fclose(fp);             //ALWAYS CLOSE THE FILE!!!


   return(0);
}
```

Notice the included error handling (fp==NULL) this checks if the file was opened successfully. It is not mandatory to do this but it might save you from a program crash. Try opening a nonexistent file like “listofpassingstudents.txt”.

Now that you have mastered the read/write from file functions you should try some exercises on your own.

Exercises:

1. Create a program that counts the number of characters in a file and displays the result in the console (on the screen).
2. Create a program that reads two matrices from a file, sums them and prints the resulting matrix to a second file. Use the filenames “matrix.in” and “matrix.out”. You can read and edit any text file with the notepad application.
3. Create a program that reads a filename from the keyboard. If the filename exists then it reads the entire file and displays the content in the console (on the screen). If it does not exist it displays “file not found” in the console window.