LAB10 : Files

Objectives:

1. To use files as input and output for programs.
2. To use while loop with file I/O.

Definitions:

File is a place on disk where a group of related data is stored.

Why files are needed?

When the program is terminated, the entire data is lost in C programming. If you want to keep large volume of data, it is time consuming to enter the entire data. But, if file is created, these information can be accessed using few commands.
There are large numbers of functions to handle file I/O in C language.

Theory:

For C File I/O you need to use a FILE pointer, which will let the program keep track of the file being accessed. (You can think of it as the memory address of the file or the location of the file).

For example:

```
        FILE *fp;
```

fopen

To open a file you need to use the fopen function, which returns a FILE pointer. Once you've opened a file, you can use the FILE pointer to let the compiler perform input and output functions on the file.

```
        fp = fopen("filename", "mode");
         fopen_s(&fp,"filename","mode");
```

where:

- filename is a string that holds the name of the file on disk
- mode is a string representing how you want to open the file. Most often you'll open a file for reading ("r") or writing ("w").

Note that fopen() returns a FILE * that can then be used to access the file. When the file cannot be opened (e.g., we don't have permission or it doesn't exist when opening for reading), fopen() will return NULL.

fopen modes

The allowed modes for fopen are as follows:

```
r  - open for reading
w  - open for writing (file need not exist)
a  - open for appending (file need not exist)
r+ - open for reading and writing, start at beginning
w+ - open for reading and writing (overwrite file)
a+ - open for reading and writing (append if file exists)
```

Note that it's possible for fopen to fail even if your program is perfectly correct: you might try to open a file specified by the user, and that file might not exist (or it might be write-protected). In those cases, fopen will return 0, the NULL pointer.

Here's a simple example of using fopen:

```
FILE *fp;
fp=fopen("c:test.txt", "r");
fopen_s(&fp,″ c:test.txt", "r");
```

This code will open test.txt for reading in text mode.

fclose

When you're done working with a file, you should close it using the function

fclose returns zero if the file is closed successfully.

An example of fclose is

```
fclose(fp);
```

Reading and writing with fprintf, fscanf, fputc, and fgetc

To work with text input and output, you use fprintf and fscanf, both of which are similar to their friends printf and scanf except that you must pass the FILE pointer as first argument. For example:

```
FILE *fp;
fp=fopen("c:test.txt", "w");
fprintf(fp, "Testing...\n");
```

It is also possible to read (or write) a single character at a time--this can be useful if you wish to perform character-by-character input (for instance, if you need to keep track of every piece of punctuation in a file it would make more sense to read in a single character than to read in a string at a time.) The fgetc function, which takes a file pointer will let you read a single character from a file:

The fputc function allows you to write a character at a time--you might find this useful if you wanted to copy a file character by character. It looks like this:

The second argument is the file to write to. On success, fputc will return the value c, and on failure, it will return EOF.

Assignment:

1) Opening a file

```
inlist.txt
----------
70
98
...
```

```
#include "stdafx.h"
#include "stdlib.h"
void main()
{
  FILE *ifp;

  fopen_s("inlist.txt", "r");

  if (!ifp) {
        printf("File opening failed");
        return EXIT_FAILURE;
  }
}
```

Note that the input file that we are opening for reading ("r") must already exist.

2) Reading from or writing to a file

Once a file has been successfully opened, you can read from it using fscanf() or write to it using fprintf(). These functions work just like scanf() and printf(), except they require an extra first parameter, a FILE * for the file to be read/written.

Continuing our example from above, suppose the input file consists of lines with an *integer test score*, e.g.

```
inlist.txt
----------
70
98
...
```

the output file we are opening for writing ("w") does not have to exist. If it doesn't, it will be created. If this output file does already exist, its previous contents will be thrown away (and will be lost).

We might use the files we opened above by copying each score from the input file to the output file. In the process, we'll increase each score by 10 points for the output file:

The function fscanf(), like scanf(), normally returns the number of values it was able to read in. However, when it hits the end of the file, it returns the special value EOF. So, testing the return value against EOF is one way to stop the loop.

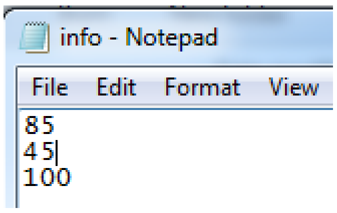
```
#include "stdafx.h"
#include "stdlib.h" //for EXIT_FAILURE

int main()
{
       FILE *ifp, *ofp;
       int score;

       fopen_s(&ifp,"inlist.txt", "r");
       fopen_s(&ofp,"outlist.txt", "w");

       if (!ifp) {
               printf("File opening failed");
               return EXIT_FAILURE;
       }

       if (!ofp) {
               printf("File opening failed");
               return EXIT_FAILURE;
       }
        // fscanf(ifp, "%d", &score) ;
        // while (!feof(ifp))

       while (fscanf_s(ifp, "%d", &score) != EOF)
       {
               fprintf(ofp, "%d\n", score + 10);
       }
      _fcloseall();

}
```

Task1: Write a program that prompts the user to input his/her name and surname into a "lower.txt" file and then convert them to upper and write them into a "upper.txt file"

lower.txt	upper.txt
alexsandro de souza	ALEXSANDRO DE SOUZA

Task2: Write a program which copies the content of a file "prog.txt" to "prognew.txt"

Task3: Write a program that reads midterm results from a file, and print midterm results of those students who got more than 50 on the SCREEN. Assume there are unknown number of records in the file.



```
85
45
100
```