C Programs: File Input/Output

File I/O in C

In C, the input can be read from the `standard input device`, (`stdin`) or – from a file residing on disk. Similarly, the output can be sent to the `standard output device` (`stdout`), or to a disk file.

The C functions for `stdin`/`stdout` I/O are `scanf()` and `printf()`. Similar functions exist for file I/O.

Stream processing. C treats input and output as **streams of data**. Input streams of data are used to read data in, output streams of data are used to print data to. A disk file to be read from/written to is also treated as a stream.

stdio.h revisited. All file I/O functions are declared in `stdio.h` header file.

Opening and closing files. To open a file for either reading or writing, use the `fopen()` function:

```
FILE * fopen(const char fileName[], const char mode[])
```

Here, `fileName[]` is the name of the file to be opened, and `mode` is the access mode for the file. The access modes are:

Mode	Meaning
`"r"`	Open file for reading (input stream)
`"w"`	Open file for writing; file created or truncated (output stream)
`"a"`	Open file for appending (output stream)
`"r+"`	Open file for reading and writing w/o destroying the file
`"w+"`	Open file for reading and writing; truncate the file
`"a+"`	Open file for reading and appending

FILE *. The return type of the `fopen()` function is `FILE *`. In C, values of this type represent I/O streams. Variables of type `FILE *` can be declared in the program and they will be used later in `fprintf()` and `fscanf()` functions.

To open a file and assign it to a `FILE *` variable, use the following code:

```
FILE * in;
FILE *out;

in = fopen("input.txt","r");    /* open file "input.txt" for reading */
out = fopen("output.txt","w");  /* create/overwrite file "output.txt", open it for writing*/
```

After this fragment is executed, two files will be open. `input.txt` will be open for reading its content. `output.txt` will be overwritten with an empty file, and will be ready for the output to be written to it.

fprintf() and fscanf()

To read and write from files, use `fprintf()` and `fscanf()` functions.

Both functions work similarly to `printf()` and `scanf()` respectively, except they take one more argument: `FILE *` value associated with the appropriate file.

Consider the following code:

```
FILE * in;
FILE *out;
int x;

in = fopen("input.txt","r");    /* open file "input.txt" for reading */
out = fopen("output.txt","w");  /* create/overwrite file "output.txt", open it for writing*/

fscanf(in, "%d", &x);

fprintf(out, "The number is %d\n", x);
```

After the two files are opened for reading and writing respectively, the program tries to read an integer value from the `input.txt` file and then prints it together with some text to the `output.txt` file.

Closing the file

Before your program ends, make sure you **close** all open files. This is done using the `fclose()` function:

```
int fclose(FILE * fp);
```

The function returns 0 if the file was succesfully closed.

End of File

Each file ends at some point, so it is impossible to read from a file infinitely many times. When `fscanf()` reaches the end of the file, it returns a special end-of-file marker, which, in C is represented by the symbolic constant `EOF`. `fscanf()` is declared as:

```
int fscanf(FILE *stream, const char *format, ...);
```

so, `EOF` is an `int` value.

Here is an example of reading numbers from a file until the file ends:

```
FILE * in;
int x;
int status; /* we use it to catch the return value of fscanf() */

in = fopen("input.txt","r");    /* open file "input.txt" for reading */

do {

   status = fscanf(in, "%d", &x);
   if (status != EOF) {
      printf("%d\n", x);
   }
while (status != EOF);
fclose(in);
```