

# Unix, C - File Operations

Mioara Joldes

October 7, 2008

These function prototypes are in `<stdio.h>`: the program must include `<stdio.h>`. The value `EOF` (indicates end-of-file) is defined in `<stdio.h>`. Its value is usually `-1`, but this should not be relied upon.

The data type `FILE` is a struct type defined in `<stdio.h>`. A stream is a pointer-to-`FILE`, and comprises a buffer plus a file. The streams `stdin`, `stdout` and `stderr` are automatically opened when a program including `<stdio.h>` is run. Unless redirected, `stdin` is connected to the keyboard, `stdout` and `stderr` to the screen. When program execution ends, all open streams are automatically closed.

## 1 Opening a file

- Syntax

```
FILE *fopen(const char *filename, const char *mode);
```

- You should include: `<stdio.h>`
- This opens the file `filename`.
- The mode `mode` can be `"r"` for reading, `"w"` for writing, or `"a"` for appending.
- The function **returns**: a stream, or `NULL` if the file cannot be opened.
- Example:

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

## 2 Closing a file

- Syntax

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

- You should include: `<stdio.h>`
- Parameters: `.stream` is closed. For an output stream, any buffered data is first written. For an input stream, any unread data is discarded.
- The function **returns**: zero, or `EOF` for an error.

## 3 Reading from a file

1. • Syntax

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

- You should include: `<stdio.h>`
- This acts exactly like `scanf` except that the data is read from `stream`, rather than `stdin`.
- The function **returns**: the number of input items converted and assigned, or `EOF` for an error.

2. • Syntax

```
int fgetc(FILE *stream);
```

- You should include: `<stdio.h>`
- This acts exactly like `getc` except that the data is read from `stream`, rather than `stdin`.

- The function **returns**: the next character read from stream (as an int), or EOF for an error. (The char is converted to an int because EOF is usually -1, while unsigned char is usually from 0 to 255 so could not be used to return EOF.)

3. • Syntax

```
char *fgets(char *s, int n, FILE *stream);
```

- You should include: `<stdio.h>`
- This reads a maximum of  $n - 1$  characters into the array `s`. It stops if a newline `'\n'` is found. The newline is included in the array. The array is terminated by the null character `'\0'`.
- The function **returns**: a pointer-to-char, or NULL for an error.

## 4 Writing to a file

1. • Syntax

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

- You should include: `<stdio.h>`
- This acts exactly like *scanf* except that the data is written to `stream`, rather than `stdout`.
- The function **returns**: the number of characters written, or a negative value for an error.
- Example:

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

2. • Syntax

```
int fputc(int c, FILE *stream);
```

- You should include: `<stdio.h>`
- This writes the character `c` (converted to `unsigned char`) to `stream`.
- The function **returns**: `c`, or EOF for an error.

3. • Syntax

```
int fputs(const char *s, FILE *stream);
```

- You should include: `<stdio.h>`
- This writes string `s` to `stream`. (Note: unlike `puts`, `fputs` does not add a newline `'\n'` to the end of `s`.)
- The function **returns**: a non-negative value, or EOF for an error.