

# Input/Output Part II

CIT 593

CIT593

1/16

## File I/O in C

A **file** is a sequence of ASCII characters stored on some device.

- Allows us to process large amounts of data without having to type it in each time or read it all on the screen as it scrolls by.

We use "**FILE**" structure to do file I/O

FILE structure has information such as:

- Location of the file
- Whether the file is being read or written
- Current character position in the file
- Whether errors or end of file has occurred

Hence we declare file handler or pointer to do any operations

```
FILE *infile;
```

Note: The **FILE** struct is defined in <stdio.h>.

CIT593

2/16

## fopen

The fopen() function opens the file in a particular mode

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

First argument: **name**

- The name of the physical file, or how to locate it on the storage device. This may be dependent on the underlying operating system

Second argument: **mode**

- How the file will be used:
  - "r" -- read from the file
  - "w" -- write, starting at the beginning of the file
  - "a" -- write, starting at the end of the file (append)
  - "rb" -- to open an existing binary file for reading
- Returns a pointer of type File
  - > If pointer is NULL (0) then error has occurred in reading the file
  - > Else a non-zero value is returned

CIT593

3/16

## Example of fopen

Opening file in.txt to read

```
FILE * inputfile = fopen("in.txt", "r");
```

Opening file out.txt to write

```
FILE * outputfile = fopen("out.txt", "w");
```

CIT593

4/16

## Reading and Writing a file

Once a file is opened, it can be read or written using `fscanf()`, `fprintf()`, `fgetc()` respectively

These are just like `scanf()`, `printf()`, `getc()` except an additional argument specifies a file pointer

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

```
FILE *infile;
FILE *outfile;
...
fprintf(outfile, "The answer is %d\n", x);
fscanf(infile, "%d", &number);
```

CIT593

5/16

## fgetc

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

- Get the next character from file stream
- Returns the next character of the file stream and increases the file pointer to point to the following one
  - The character read is returned as an **int** value.
- If the End Of File has been reached or there has been an error reading, the function returns EOF
  - EOF is a macro, usually -1, as ASCII range is 0 to 127

CIT593

6/16

## fclose

**int fclose(FILE \*stream)**

- The function closes the file associated with the stream .
- It returns zero if successful; otherwise, it returns EOF.
- All internal buffers associated with the stream are flushed
  - The content of any unwritten buffer is written and the content of any unread buffer is discarded
- Even if the call fails, the stream passed as parameter will no longer be associated with the file

CIT593

7/16

## Example of File I/O: textcopy.c

```
#include <stdio.h>

int main(int argc, char * argv[ ] )
{
    char * inputname;
    char * outputname;

    FILE * infile;
    FILE * outfile;

    int charread = 0;
    if (argc != 3)
    {
        printf("Usage: textcopy inputfile\noutputfile.\n");
        return 1;
    }
    inputname = argv[1];
    outputname = argv[2];

    /* Open/ create files. */
    infile = fopen(inputname, "r"); /* mode "Read" */
    outfile = fopen(outputname, "w"); /* mode "Write" */

    if (infile == NULL || outfile == NULL) /* files did not open */
    {
        printf("Files could not be opened\n");
        return 1; /* quit now */
    }

    while ((charread = fgetc(infile)) != EOF)
    {
        printf("%c", charread);
        fprintf(outfile, "%c", charread);
    }

    /* close the file */
    if (fclose(infile) != 0 || fclose(outfile) != 0)
    {
        printf("Close file error.");
    }
    return 0;
}
//end if main
```

Reads file & gets a character from file till EOF is encountered

CIT593

8/16

## Standard Error (stderr)

### Standard error

- Is another output stream typically used by programs to output error messages or diagnostics
- It is a stream independent of standard output and can be redirected separately
- The usual destination is the text terminal
  - Note: Redirecting standard output does not redirect standard error

Example:

```
fprintf(stderr, "Can't open input file!\n");
```

CIT593

9/16

## exit(), EXIT\_SUCCESS, EXIT\_FAILURE

### void exit(int status)

- Terminates a program/process
- Flushes all output streams, closes all open streams
- *status* value is 0 to 255
  - There are conventions for what sorts of status values certain programs should return
  - The most common convention is simply 0 for success and 1 for failure

### EXIT\_FAILURE

- Unsuccessful termination for *exit()*
- Non-zero value

### EXIT\_SUCCESS

- Successful termination for *exit()*
- Zero value


Include `stdlib.h` to use the above

CIT593

10/16

## Carriage Return (CR) control character

### In mechanical typewriters with levers

- The lever allows the printing head to return to the left side of the paper after a line of text had been typed
- Adopted in electronic typewriters and keyboards as **CR** character. Keys generating carriage return are keys with the following symbol/words:
  - Enter 
  - Return
- ASCII code for CR is 13 in decimal or 0D in hex

CIT593

11/16

## Line Feed (LF) control character

### In a mechanical typewriter

- A dial/knob advances the paper one line without moving the print head

### In electronic typewriter or keyboards

- It is the **LF** control character that moves the position of the cursor to the next line

ASCII value for LF is 10 (in dec) or 0A(in hex)

**CR+LF** = used as line termination sequence

Window OS uses **CR+LF** to indicate line termination in the text files

CIT593

12/16

## Newline (“\n”)

In Unix/Linux OS, line feed is used as line termination sequence

- Implicit CR before LF is a unix invention, probably as an economy, since it saves one byte per line.

C language uses LF to terminate lines and called it the **newline “\n”** for convenience or readability. Hence ascii value for \n is same as LF i.e. x0A or 10

CIT593

13/16

## About Files

Ultimately every thing in the computer is stored as 0's & 1's

Types of files:

1. Text files can be displayed, edited by regular file editors

- Contain only ASCII characters
- Divided into lines; end of line marked by a control character e.g. CR, LF, or CR+LF
- a C source file is an example of a text file

2. Non-text files

- Do not print or display with regular text editor
- ASCII + other information (pics, diagrams etc)
  - > 1. Formatted files such as pdf, doc, ppt
  - > 2. Binary & executable files e.g. a.out
- Contain arbitrary bit patterns

CIT593

14/16

## Line Termination in Text Files

UNIX text files

- End of line marked by LF

Windows text files

- End of line marked by CR followed by LF

Macintosh text files

- End of line marked by CR

Note:

- Variation in line termination can be problematic when exchanging data between systems of different types.

CIT593

15/16

## Line Termination Solution

Use editors that can handle files

- With whatever line endings you have or want
- E.g. Notepad++

Write your own tools

- Dave Matuszek's fixline.jar
  - > Fixes line ending based on the system

CIT593

16/16