C I/O

Print/Scan Formats

- `%c`: single character
- `%d`: signed decimal integer
- `%i`: signed decimal integer
- `%e`: floating-point number (scientific notation)
- `%E`: floating-point number (scientific notation)
- `%f`: floating-point number (decimal format)
- `%G`: floating-point number that generates fewest characters (scientific, decimal format)
- `%o`: unsigned octal character
- `%s`: C-string printed until terminating null
- `%u`: unsigned decimal integer
- `%x`: unsigned hex number
- `%X`: unsigned hex except uses uppercase letters
- `%p`: pointer or address value
- `%%`: literal %

Advanced `printf` format syntax

`% [flags] [width] [.precision] [length_char] specifier`

Flags

- `+` +/- sign printed for signed values
- `-` left-justifies the output
- `#` octal prefix or hex prefix to be printed
- *space* space printed if no +/- sign is going to be printed
- `0` field padded with `0`s instead of spaces

Width: width of the field

Precision: number of digits printed (for floating point values)

length_char

- `h` short integer format
- `l` long integer format
- `L` long double format

`scanf` formats

- `%c` character
- `%d` digit as decimal integer
- `%f/e/E/g/G` floating point
- `%o` octal digit
- `%s` string, read until whitespace encountered
- `%u` unsigned decimal

- `%x/X` hex

`%[*][width][length_char]specifier`

asterisk indicated data is to be read but ignored

width max number of characters to be read for this field

length_char size of the data type involved

- `h` short
- `l` long
- `L` long double

Input and Output to Strings

`sprintf` and `sscanf` can be used to write to a string directly.

File I/O

1. Declare a file pointer `FILE *fp`
2. Get file pointer by calling `fopen`
3. Use file pointer to read/write to the file
4. Close the file `fclose`

Opening a file

`fopen(filename, mode`

Modes:

- `r`: read-only
- `w`: write, if exists contents are erased
- `a`: append
- `r+`: read/write, file must exist
- `w+`: read/write, if exists contents are erased
- `a+`: read/write, previous contents protected
- `t`: (def) text mode
- `b`: binary mode, can be used as a modifier on all previous modes

Closing a file

Closing a file is good practise, open files may not be accessible to other programs. Changes are realised upon closing.

`fclose(file_ptr)`

Reading and Writing Text Files

- `fputs(str, file_ptr)`: writes `str` to file

- `fgetc(file_ptr)`: returns the next character from the specified input stream
- `fgets(file_ptr)`: reads characters from input stream until a newline or EOF
- `fprintf(file_ptr, format_str, [,args])`: writes formatted output to output stream
- `fputc(ch, file_ptr)`: writes char to file
- `fputs(str, file_ptr)`: writes a C-string to file
- `fscanf(file_ptr, format_str, [,args])`: reads texts interpreted according to format specified
- `getc(file_ptr)`: returns the next character from the specified input
- `putc(ch, file_ptr)`: writes `ch` to specified file

Reading and Writing Binary Files

`reinterpret_cast<char*>` operator should be used.

Random-Access Function

Random-access functions move the file position indicator, by default the pointer moves sequentially.

- `fgetpos(file_ptr, fpos_ptr)`: saves the current file pointer into an object of type `fpos_t`
- `fsetpos(file_ptr, fpos_ptr)`: restores a saved position
- `fseek(file_ptr, offset, origin)`: sets the file position indicator to an offset from a specified origin -`ftell(file_ptr)`: returns the file-position indicator for the specified stream, returned as a `long int`

Other File-management functions

- `clearerr(file_ptr)`: clears the error status of a stream
- `ferror(file_ptr)`: returns non-zero value if there is an error
- `fflush(file_ptr)`: flushes the input or output buffer - reads and writes all the data in the buffer
- `freopen(filename_str, mode_str, file_ptr)`: attempts to close stream and reassigns the stream to the named file
- `remove(filename_str)`: deletes the named file from disk
- `rename(oldname_str, newname_str)`: renames the *oldname* to *newname*
- `rewind(file_ptr)`: reset the position indicator to the beginning of the specified stream
- `tempfile()`: returns a file-pointer that can be used as a temporary file
- `tempname(str)`: returns an available temporary file name as a C-string