CSD101: Introduction to computing and programming (ICP)

Streams

- A **stream** in C is the abstraction that is used for input-output.
- A program uses an input stream when data has to be read into a program and uses an output stream when data has to be written out.
- We have already encountered 3 streams.

  stdin : Standard input stream associated with the keyboard.
  stdout : Standard output stream associated with the terminal.
  stderr : Standard error stream also associated with the terminal.

- `scanf` reads from `stdin`, `printf` writes to `stdout`, error messages are sent to `stderr`.

Files

- A file is a sequence of bytes, typically stored on a disk. A file also has a name. The type of data stored could be text (i.e. characters) or binary. A file descriptor describes attributes of a file like its type, size, name, date it was created etc.
- Files are entities managed by the operating system but can be connected to programs as input/output streams when needed.

Files in **C** I

- In **C** -89 all input-output is handled by the `stdio` library. The prototypes of the functions and some constants that are used are in the header `stdio.h`.
- **C** has a `FILE` type and files are referenced using FILE pointers.
- The function that creates an object of type FILE and returns a pointer to it is: `fopen`. Its prototype is:
  `FILE *fopen(const char *filename, const char *mode)`
  The `mode` can be:
  "r" - opens for reading.
  "w" - opens for writing from beginning, a file is created if it does not exist.
  "a" - append starts writing at the end of the file if file exists else creates a new file.

Files in C II

"r+" - opens for reading and writing.
"w+" - opens for reading and writing, first truncates file to 0 size or creates it if not present.
"a+" - opens for both reading, writing; Reading starts at beginning, writing is only at the end.

- If the file has binary data then a 'b' must be added after the first character of the mode. For example: "rb+" for reading, writing. "r+b" will also work.
- After use a file must be closed.
  `int fclose(FILE *fp)` will close the file associated with `fp` and returns 0 if successful and -1 or EOF if it fails.

Reading a file - basic functions

- The functions to read from a file correspond to those that read from `stdin`, with an 'f' as the first character.
  - `int fgetc(FILE *fp)` - reads a character from stream `fp`. The return value is the character read or -1 (EOF or NULL) if end of file is reached or there is an error.
  - `char *fgets(char *buf, int n, FILE *fp)` - reads $n-1$ chars from `fp` into `buf` and appends a NULL character at the end. If it encounters a `'\n'` or `EOF` then it returns only chars till that point including the newline char.
  - `int fscanf(FILE *fp, const char *format,...)` - works exactly like `scanf`. Actually, `scanf` is equivalent to `fscanf(stdin, ...)`.
- `stdio` has more functions for input. See file `stdio.h` for the prototypes and the texts for details of how the functions behave.

Writing to a file - basic functions

- The functions that write to a file correspond to functions that write to `stdout`. They have 'f' as the first character.
  - `int fputc(int c, FILE *fp)` - writes the character value of `c` to the stream `fp`. If there is an error it will return -1 (EOF or NULL) else returns 0.
  - `int fputs(const char *s, FILE *fp)` - writes the string `s` to the output stream `fp`. If there is an error it returns -1 (EOF or NULL).
  - `int fprintf(FILE *fp, const char *format, ...)` - works exactly like `printf`. `printf` is equivalent to `fprintf(stdout, ...)`.
- More output functions are available. See `stdio.h` for the prototypes and the texts for documentation.

I/O redirection

- One way to read from a file or write to a file is by using I/O redirection. This is a feature of the OS.
- Let `exe` be an executable file that reads from `stdin` and writes to `stdout`.
  To read from file `in.txt` and write to file `out.txt`. Run at the command line with redirection operators '<' and '>':
  `./exe < in.txt > out.txt`

Command line arguments

- Arguments can be given on the command line.
- The `main` function can take two arguments `(int argc, char *argv[])`. The names could be anything but traditionally they are called `argc` and `arg[]`.
- `argc` gives the number of arguments including the name of the executable and `argv[]` is an `argc` sized array of strings where *argv*[0] is the executable program string and `argv[1]` to `argv[n-1]` are the command line arguments represented as strings.
- So, file names can be passed in via command line arguments.