Summary Page: File I/O Using the C Library
CISC 220, fall 2012

Opening a File:

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

`mode` can be: `"r"` (read), `"w"` (write), `"a"` (append)

`fopen` will return `NULL` and set `errno` if file can't be opened

Closing a File:

`fclose(FILE* file)`

Predefined File Pointers:

`stdin`: standard input

`stdout`: standard output

`stderr`: standard error

Reporting Errors:

`char *strerror(int errnum)`: Returns a string describing an error.

`void perror(char *msg)`: Prints an error message based on current value of `errno`, with `msg` as a prefix

Character Input:

`int getc(FILE *stream)`: reads a character and returns it (or EOF if at end of file)

`int getchar()`: equivalent to `getc(stdin)`

`ungetc(int c, FILE *stream):` "pushes" `c` back onto input stream

Character Output:

`putc(int c, FILE *stream):` writes `c` to the file

`putchar(c)`: equivalent to `putc(stdout)`

String Output:

`fputs(char *s, FILE *stream)`: writes `s` to the file

`puts(char *s)`: writes s *plus '\n'* to `stdout`

String Input:

`char* fgets (char *s, int count, FILE *stream)`

Reads characters from stream until end of line OR count-1 characters are read.

Will include an end of line character (`'\n'`) if it reaches the end of the line

On return, `s` will always have a null character (`'\0'`) at the end.

Returns `NULL` if we're already at the end of file or if an error occurs.

Formatted I/O:

`fscanf(FILE *stream, char *format,` *more args...*`)`: Works like `scanf`, but reads from the specified file.

`fprintf(FILE *stream, char *format,` *more args...*`)`: Works like `printf`, but writes to the specified file