Reference Material from K&R

B1.4 Character Input and Output Functions

`int fgetc(FILE *stream)`
`fgetc` returns the next character of `stream` as an `unsigned char` (converted to an `int`), or `EOF` if end of file or error occurs.

`char *fgets(char *s, int n, FILE *stream)`
`fgets` reads at most the next `n-1` characters into the array `s`, stopping if a newline is encountered; the newline is included in the array, which is terminated by `'\0'`. `fgets` returns `s`, or `NULL` if end of file or error occurs.

`int fputc(int c, FILE *stream)`
`fputc` writes the character `c` (converted to an `unsigned char`) on `stream`. It returns the character written, or `EOF` for error.

`int fputs(const char *s, FILE *stream)`
`fputs` writes the string `s` (which need not contain `'\n'`) on `stream`; it returns non-negative, or `EOF` for an error.

`int getc(FILE *stream)`
`getc` is equivalent to `fgetc` except that if it is a macro, it may evaluate `stream` more than once.

`int getchar(void)`
`getchar` is equivalent to `getc(stdin)`.

`char *gets(char *s)`
`gets` reads the next input line into the array `s`; it replaces the terminating newline with `'\0'`. It returns `s`, or `NULL` if end of file or error occurs.

`int putc(int c, FILE *stream)`
`putc` is equivalent to `fputc` except that if it is a macro, it may evaluate `stream` more than once.

`int putchar(int c)`
`putchar(c)` is equivalent to `putc(c,stdout)`.

`int puts(const char *s)`
`puts` writes the string `s` and a newline to `stdout`. It returns `EOF` if an error occurs, non-negative otherwise.

`int ungetc(int c, FILE *stream)`
`ungetc` pushes `c` (converted to an `unsigned char`) back onto `stream`, where it will be returned on the next read. Only one character of pushback per stream is guaranteed. `EOF` may not be pushed back. `ungetc` returns the character pushed back, or `EOF` for error.

B2. Character Class Tests: <ctype.h>

The header <ctype.h> declares functions for testing characters. For each function, the argument is an int, whose value must be EOF or representable as an unsigned char, and the return value is an int. The functions return non-zero (true) if the argument c satisfies the condition described, and zero if not.

isalnum(c)	isalpha(c) or isdigit(c) is true
isalpha(c)	isupper(c) or islower(c) is true
iscntrl(c)	control character
isdigit(c)	decimal digit
isgraph(c)	printing character except space
islower(c)	lower-case letter
isprint(c)	printing character including space
ispunct(c)	printing character except space or letter or digit
isspace(c)	space, formfeed, newline, carriage return, tab, vertical tab
isupper(c)	upper-case letter
isxdigit(c)	hexadecimal digit

In the seven-bit ASCII character set, the printing characters are 0x20 (' ') to 0x7E ('~'); the control characters are 0 (NUL) to 0x1F (US), and 0x7F (DEL).

In addition, there are two functions that convert the case of letters:

int tolower(int c)	convert c to lower case
int toupper(int c)	convert c to upper case

If c is an upper-case letter, tolower(c) returns the corresponding lower-case letter; otherwise it returns c. If c is a lower-case letter, toupper(c) returns the corresponding upper-case letter; otherwise it returns c.

B3. String Functions: <string.h>

There are two groups of string functions defined in the header <string.h>. The first have names beginning with str; the second have names beginning with mem. Except for memmove, the behavior is undefined if copying takes place between overlapping objects. Comparison functions treat arguments as unsigned char arrays.

In the following table, variables s and t are of type char *; cs and ct are of type const char *; n is of type size_t; and c is an int converted to char.

char *strcpy(s,ct)	copy string ct to string s, including '\0'; return s.
char *strncpy(s,ct,n)	copy at most n characters of string ct to s; return s. Pad with '\0's if t has fewer than n characters.
char *strcat(s,ct)	concatenate string ct to end of string s; return s.
char *strncat(s,ct,n)	concatenate at most n characters of string ct to string s, terminate s with '\0'; return s.
int strcmp(cs,ct)	compare string cs to string ct; return <0 if cs<ct, 0 if cs==ct, or >0 if cs>ct.
int strncmp(cs,ct,n)	compare at most n characters of string cs to string ct; return <0 if cs<ct, 0 if cs==ct, or >0 if cs>ct.
char *strchr(cs,c)	return pointer to first occurrence of c in cs or NULL if not present.
char *strrchr(cs,c)	return pointer to last occurrence of c in cs or NULL if not present.

`size_t strspn(cs,ct)`	return length of prefix of `cs` consisting of characters in `ct`.
`size_t strcspn(cs,ct)`	return length of prefix of `cs` consisting of characters *not* in `ct`.
`char *strpbrk(cs,ct)`	return pointer to first occurrence in string `cs` of any character of string `ct`, or `NULL` if none are present.
`char *strstr(cs,ct)`	return pointer to first occurrence of string `ct` in `cs`, or `NULL` if not present.
`size_t strlen(cs)`	return length of `cs`.
`char *strerror(n)`	return pointer to implementation-defined string corresponding to error `n`.
`char *strtok(s,ct)`	`strtok` searches `s` for tokens delimited by characters from `ct`; see below.

A sequence of calls of `strtok(s,ct)` splits `s` into tokens, each delimited by a character from `ct`. The first call in a sequence has a non-`NULL` `s`. It finds the first token in `s` consisting of characters not in `ct`; it terminates that by overwriting the next character of `s` with `'\0'` and returns a pointer to the token. Each subsequent call, indicated by a `NULL` value of `s`, returns the next such token, searching from just past the end of the previous one. `strtok` returns `NULL` when no further token is found. The string `ct` may be different on each call.

The `mem...` functions are meant for manipulating objects as character arrays; the intent is an interface to efficient routines. In the following table, `s` and `t` are of type `void *`; `cs` and `ct` are of type `const void *`; `n` is of type `size_t`; and `c` is an `int` converted to an `unsigned char`.

`void *memcpy(s,ct,n)`	copy `n` characters from `ct` to `s`, and return `s`.
`void *memmove(s,ct,n)`	same as `memcpy` except that it works even if the objects overlap.
`int memcmp(cs,ct,n)`	compare the first `n` characters of `cs` with `ct`; return as with `strcmp`.
`void *memchr(cs,c,n)`	return pointer to first occurrence of character `c` in `cs`, or `NULL` if not present among the first `n` characters.
`void *memset(s,c,n)`	place character `c` into first `n` characters of `s`, return `s`.