B1.6 File Positioning Functions

`int fseek(FILE *stream, long offset, int origin)`
`fseek` sets the file position for `stream`; a subsequent read or write will access data beginning at the new position. For a binary file, the position is set to `offset` characters from `origin`, which may be `SEEK_SET` (beginning), `SEEK_CUR` (current position), or `SEEK_END` (end of file). For a text stream, `offset` must be zero, or a value returned by `ftell` (in which case `origin` must be `SEEK_SET`). `fseek` returns non-zero on error.

`long ftell(FILE *stream)`
`ftell` returns the current file position for `stream`, or `-1L` on error.

`void rewind(FILE *stream)`
`rewind(fp)` is equivalent to `fseek(fp,0L,SEEK_SET); clearerr(fp)`.

`int fgetpos(FILE *stream, fpos_t *ptr)`
`fgetpos` records the current position in `stream` in `*ptr`, for subsequent use by `fsetpos`. The type `fpos_t` is suitable for recording such values. `fgetpos` returns non-zero on error.

`int fsetpos(FILE *stream, const fpos_t *ptr)`
`fsetpos` positions `stream` at the position recorded by `fgetpos` in `*ptr`. `fsetpos` returns non-zero on error.

B1.7 Error Functions

Many of the functions in the library set status indicators when error or end of file occur. These indicators may be set and tested explicitly. In addition, the integer expression `errno` (declared in `<errno.h>`) may contain an error number that gives further information about the most recent error.

`void clearerr(FILE *stream)`
`clearerr` clears the end of file and error indicators for `stream`.

`int feof(FILE *stream)`
`feof` returns non-zero if the end of file indicator for `stream` is set.

`int ferror(FILE *stream)`
`ferror` returns non-zero if the error indicator for `stream` is set.

`void perror(const char *s)`
`perror(s)` prints `s` and an implementation-defined error message corresponding to the integer in `errno`, as if by

```
fprintf(stderr, "%s: %s\n", s, "error message")
```

See `strerror` in Section B3.

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.

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`.

B4. Mathematical Functions: <math.h>

The header `<math.h>` declares mathematical functions and macros.

The macros `EDOM` and `ERANGE` (found in `<errno.h>`) are non-zero integral constants that are used to signal domain and range errors for the functions; `HUGE_VAL` is a positive `double` value. A *domain error* occurs if an argument is outside the domain over which the function is defined. On a domain error, `errno` is set to `EDOM`; the return value is implementation-dependent. A *range error* occurs if the result of the function cannot be represented as a `double`. If the result overflows, the function returns `HUGE_VAL` with the right sign, and `errno` is set to `ERANGE`. If the result underflows, the function returns zero; whether `errno` is set to `ERANGE` is implementation-defined.

In the following table, `x` and `y` are of type `double`, `n` is an `int`, and all functions return `double`. Angles for trigonometric functions are expressed in radians.