Standard Library Functions – Part One

Zachary Bennett
Software Engineer

@z_bennett_ zachbennettcodes.com

Header Files

General Functions

stdlib.h

Input/Output

stdio.h

Variable Arguments

stdarg.h

Time Functions

time.h

Locale Functions

locale.h

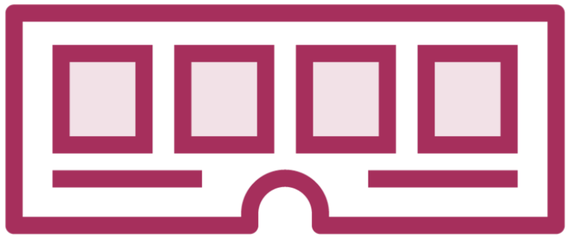
Math Functions

math.h

General Library Functions – stdlib.h

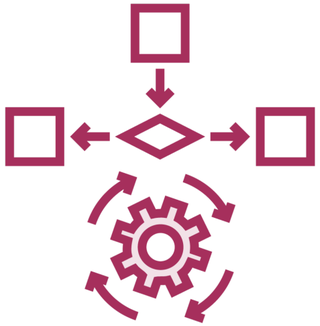
Sections

Conversion

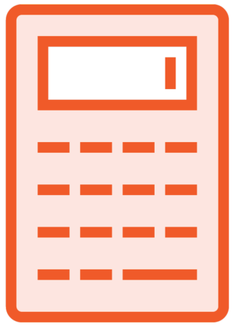


Memory

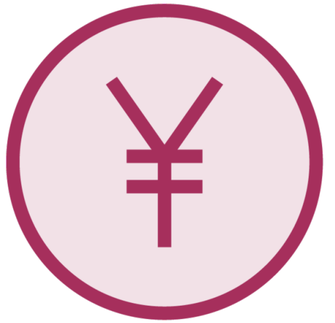
Process/Program



Algorithm



General Math



Multi-byte Character

Conversion Functions

Function Signature	Purpose
`int atoi(const char* str)`	Converts a C string to an integer
`long int atol(const char* str)`	Converts a C string to a long integer
`double atof(const char* str)`	Converts a C string to a double
`double strtod(const char* str, char** outptr)`	Converts the first number found within a C string to a double and sets the outptr to the rest of the string after conversion
`long int strtol(const char* str, char** outptr, int base)`	Converts the first number found within a C string to a long integer in the right base and sets the outptr to the rest of the string after the conversion.
`unsigned long int strtoul(const char* str, char** outptr, int base)`	Converts the first number found within a C string to an unsigned long integer in the right base and sets the outptr to the rest of the string after the conversion.

Memory Functions

Function Signature	Purpose
`void* malloc(size_t size)`	Allocates a chunk of memory on the heap given a size in bytes
`void* calloc(size_t count, size_t size)`	Allocates and zeroes out a chunk of memory on the heap given the size of an item in bytes and the number of items
`void* realloc(void* ptr, size_t size)`	Given a pointer to previously allocated memory, reallocates the size of the chunk of memory in question to the given size
`void free(void* ptr)`	Frees up the chunk of memory pointed to by the given pointer

Process/Program Functions

Function Signature	Purpose
void abort(void)	Causes program execution to terminate abnormally
int atexit(void (*foo) (void))	Specifies that the function “foo” will execute when the program terminates normally
void exit(int status)	Causes program execution to terminate normally
char* getenv(const char* key)	Returns the value of a process environment variable based on a given key
int system(const char* command)	Causes the host environment to execute the given command

Algorithms

Function Signature	Purpose

`void* bsearch(`
`  const void* item_to_find,`
`  const void* array,`
`  size_t array_length,`
`  size_t element_size,`
`  int (*compare)(const void *, const void *)`
`)`
	Uses the binary search algorithm to locate a given item in a chunk of memory using the given comparison function

`void qsort(`
`  const void* array,`
`  size_t array_length,`
`  size_t element_size,`
`  int (*compare)(const void *, const void *)`
`)`
	Sorts the given array using the quicksort algorithm

Math Functions

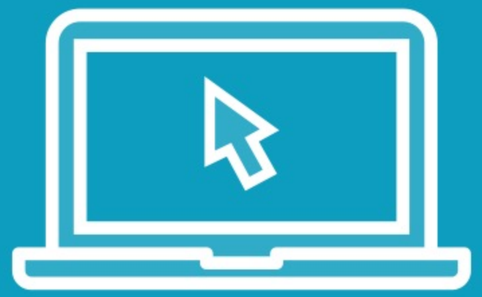
Function Signature	Purpose
int abs(int num) long int labs(long int num)	Returns the absolute value of num
div_t div(int numerator, int denominator) ldiv_t ldiv(long int numerator, long int denominator)	Divides the numerator by the denominator
int rand(void)	Returns a random number between 0 and RAND_MAX where RAND_MAX is a macro defined by the stdlib.h header file.
void srand(unsigned int seed)	This function is used in tandem with the “rand” function. This function seeds the random number generator that is used.

Multi-byte Character Utility Functions

Function Signature	Purpose
size_t mbstowcs(wchar_t* pwcs, const char* str, size_t max)	Converts the multi-byte string “str” to the wide character string “pwcs”. The ”max” parameter represents the max number of characters to convert.
int mbtowc(wchar_t* pwc, const char* str, size_t max)	Converts one multi-byte character to a wide character
int mblen(const char* str, size_t max)	Returns the length of the multi-byte character given the C string “str”. The ”max” parameter is the max number of bytes to be checked.
size_t wcstombs(char* str, const wchar_t* pwcs, size_t max)	Converts an array of wide characters to multi-byte characters
int wctomb(char* str, wchar_t* wide_char, size_t max)	Converts a wide character to a multi-byte character

Up Next:
Demo: General Library Functions

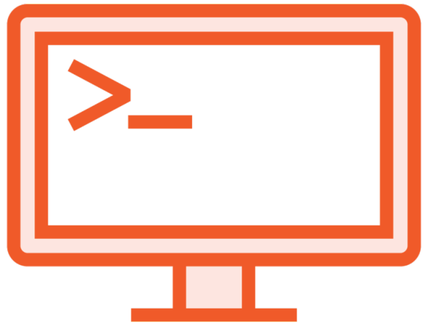
Demo



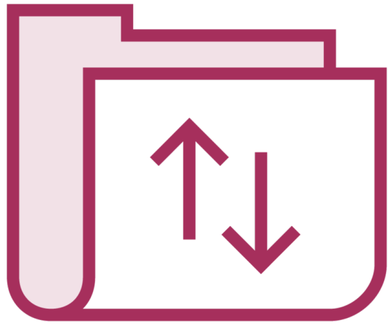
- General standard library functions
- Example function usage:
  - Conversion
  - Memory
  - Process/Program
  - Algorithms
  - Math
  - Multi-byte Utilities

Input and Output – stdio.h

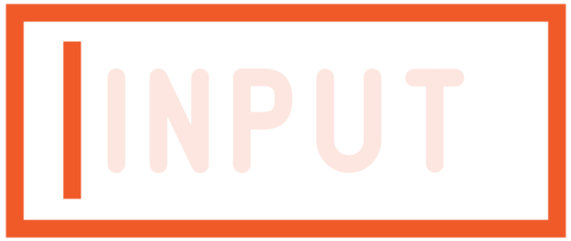
Sections



Formatted I/O



File I/O



Character I/O

File Operations

File Positioning

Error Handling

Formatted I/O

Function Signature	Purpose
int fprintf(FILE* stream, const char* format, ...)	Formatted file writing
int fscanf(FILE* stream, const char* format, ...)	Formatted file reading
int vfprintf(FILE* stream, const char* format, va_list args)	Formatted file writing with variable arg list
int printf(const char* format, ...)	Formatted writing
int scanf(const char* format, ...)	Formatted reading
int vprintf(const char* format, va_list args)	Formatted writing with variable arg list
int sprintf(char* str, const char* format, ...)	Formatted string writing
int sscanf(const char* str, const char* format, ...)	Formatted string reading
int vsprintf(char* str, const char* format, va_list args)	Formatted string writing with variable arg list

File I/O

Function Signature	Purpose

size_t fread(
  void* array_ptr,
  size_t size,
  size_t num_bytes,
  FILE* stream)
)
	Reads data from the given file stream and puts it into the array pointed to by the given void pointer

size_t fwrite(
  const void* array_ptr,
  size_t size,
  size_t num_bytes,
  FILE* stream)
)
	Writes data to the given file stream from the array pointed to by the given void pointer.

Character I/O

Function Signature	Purpose
int fgetc(FILE* stream)	Fetches the next character from the given file stream
char* fgets(char* str, int count, FILE* stream)	Fetches a line from the stream. This function stops when either “count – 1” characters are read, a newline character is reached or the EOF marker is reached.
int fputc(int char, FILE* stream)	Writes a character to the given file stream
int fputs(const char* str, FILE* stream)	Writes a C string to the given stream, excluding the null character
int getchar(void)	Fetches a character from standard input
int putchar(int char)	Writes a character to standard output
int ungetc(int char, FILE* stream)	Puts a character back onto the given stream. You can think of this as the opposite to ”fgetc”.

File Operations

Function Signature	Purpose
`int fclose(FILE* stream)`	Closes a file and flushes buffers
`FILE* fopen(const char* filename, const char* mode)`	Opens a file in the given mode
`FILE* freopen(const char* filename, const char* mode, FILE* stream)`	Reopens a closed file
`int fflush(FILE* stream)`	Flushes the buffer of the stream
`int setvbuf(FILE* stream, char* buffer, int mode, size_t size)`	Sets the buffer of a given stream
`int remove(const char* filename)`	Deletes a file
`int rename(const char* old_filename, const char* new_filename)`	Renames a file
`FILE* tmpfile(void)`	Creates a temporary file
`char* tmpnam(char* str)`	Generates a temporary file name (DEPRECATED)

File Positioning Functions

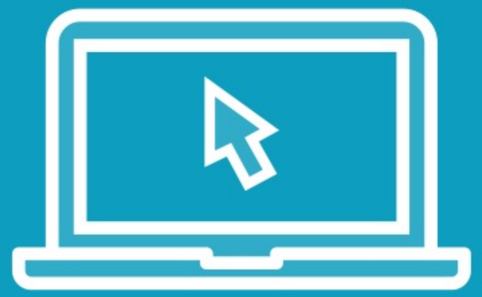
Function Signature	Purpose
int fgetpos(FILE* stream, fpos_t* pos)	Puts the current file position of the given stream into the value pointed to by the “pos” pointer
int fsetpos(FILE* stream, const fpos_t* pos)	Uses the given position to set the file position of the given stream
int fseek(FILE* stream, long int offset, int from)	Given an offset from the position of the “from” argument, this function sets the position of the file stream. The SEEK_SET, SEEK_CUR, SEEK_END macros are used with this function.
long int ftell(FILE* stream)	Fetches the file position of the given stream
void rewind(FILE* stream)	Sets the file position of the given stream to the beginning.

Error Handling Functions

Function Signature	Purpose
void clearerr(FILE* stream)	Clears any errors or EOF indicators associated with the given stream
int feof(FILE* stream)	Tests the EOF indicator for the given stream
int ferror(FILE* stream)	Tests the error indicator for the given stream
void perror(const char* str)	This function is used to print an error message after a function may have caused a system error. This function works with ”errno” and will print the string given to it followed by a colon, a space and a descriptive error that is tied to the current “errno” interpretation.

Up Next:
Demo: Input and Output Functions

Demo



- I/O functions
- Example function usage:
  - Formatted I/O
  - File I/O
  - Character I/O
  - File Operations
  - File Positioning
  - Error Handling

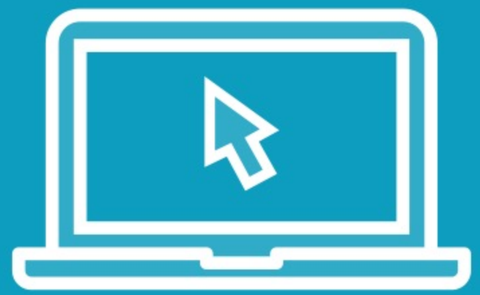
Variadic Functions – stdarg.h

Variadic Function Macros

Macro Description	Purpose
void va_start(va_list list, arg_before_list)	This macro initializes a variable argument list
type va_arg(va_list list, type)	This macro fetches the next argument in the list of the given type
void va_end(va_list list)	This macro allows the variadic function to return normally by marking that it is done with the variable argument list

Up Next:
Demo: Variadic Functions

Demo



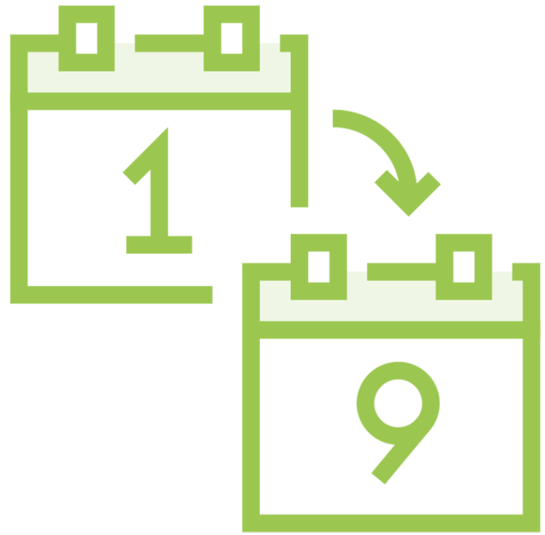
- Inspect a printf wrapper created for WBC
- See examples of the stdarg.h macros

Time Functions – time.h

Sections

Core Functions

Primary time functions which facilitate timers and clocking



Conversion/Formatting

These functions help with the formatted display and conversion of time

Core Functions

Function Signature	Purpose
clock_t clock(void)	Fetches the clock time of the computer's processor. Useful for marking certain sections of a program in order to compute execution times.
double difftime(time_t first, time_t second)	Returns the difference, in seconds, between the "first" time and the "second" time. It subtracts "second" from "first".
time_t time(time_t* timer)	Computes the current calendar time

Conversion/Formatting

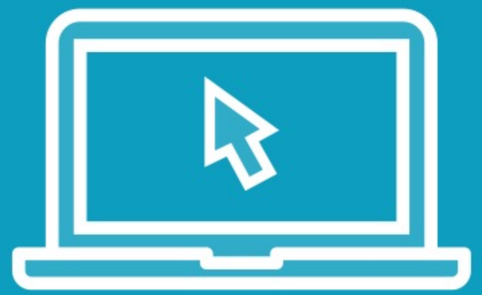
Function Signature	Purpose

size_t strftime(
  char* str,
  size_t max,
  const char* format,
  const struct tm* time_ptr
)
	Puts a formatted time string into the “str” argument given a format and a “tm” struct

time_t mktime(struct tm* time_ptr)	Given a “tm” struct, this function fetches the correct time_t value in local time
struct tm* localtime(const time_t* time)	Reverse of “mktime”
struct tm* gmtime(const time_t* time)	Like local time but uses UTC/GMT
char* ctime(const time_t* time)	Computes a C string based on a given time. The time zone used is local.
char* asctime(const struct tm* time_ptr)	Computes a C string containing the day/time given the “tm” struct

Up Next:
Demo: Time Functions

Demo



- **Time functions**
- **Local and UTC/GMT time computation**
- **Clock speed**

Locale Functions – locale.h

Locale Category Macros

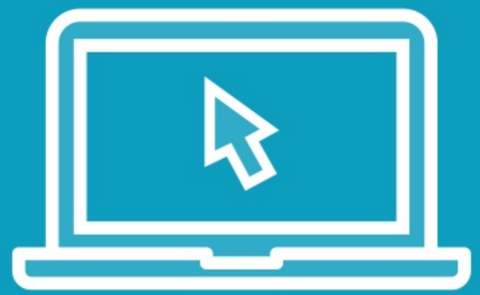
Macro	Purpose
LC_ALL	Sets all of the below macro categories to the given locale
LC_COLLATE	Sets the locale of the strcoll and strxfrm standard library functions
LC_CTYPE	Sets the locale of all of the character functions in the standard library
LC_MONETARY	Sets the locale of the money fields in the struct returned by the localeconv function
LC_NUMERIC	Sets the locale of the numeric fields and decimal formatting in the struct returned by the localeconv function
LC_TIME	Sets the locale of the strftime time formatting function in the standard library

Locale Functions

Function Signature	Purpose
char* setlocale(int category, const char* locale)	Sets values of the “lconv” struct returned by a called to “localeconv” and possibly other C library functions based on a given ”locale” string and a category. If locale is not NULL, the values/functions relevant to the given category are set to the given locale. If locale is NULL the values/functions relevant to the given category are set from environment variables that use the same names as the macro definitions defined prior to this slide.
struct lconv* localeconv(void)	Meant to be used after “setlocale” this function fetches the current locale being used by the program.

Up Next:
Demo: Locale Functions

Demo



- **Locale functions/macros**
- **Setting the locale of a program**

Math Functions – math.h

Commonly Used Math Functions

Function Signature	Purpose
double ceil(double x)	Calculates the smallest integer greater than or equal to x
double floor(double x)	Calculates the largest integer less than or equal to x
double fabs(double x)	This is the floating-point equivalent to “abs” and “labs” – functions which both reside within the stdlib.h header file
double log(double x)	Calculates the base-e logarithm of x
double pow(double x, double y)	Calculates the value of x raised to the power of y
double sqrt(double x)	Calculates the square root of x
double exp(double x)	Calculates the value of e raised to the xth power

Trigonometric functions

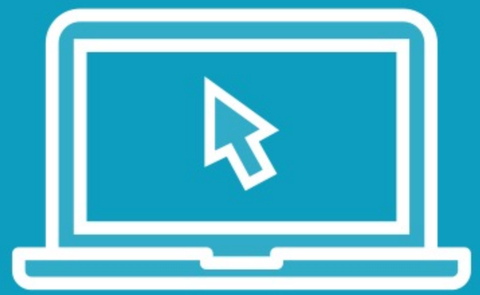
Exponential functions

Floating point functions

Honorable Mention

https://en.wikipedia.org/wiki/C_mathematical_functions

Demo



- Commonly used math functions in action
  - ceil/floor
  - fabs
  - log
  - exp
  - pow
  - sqrt

Summary

- **Standard library functions:**
  - **General**
  - **I/O**
  - **Variadic**
  - **Time**
  - **Locale**
  - **Math**

Up Next:
Standard Library Functions – Part Two