

# Standard Library Functions – Part Two

---



**Zachary Bennett**

Software Engineer

@z\_bennett\_ zachbennettcodes.com

# Header Files

## String Functions

string.h

## Character Testing

ctype.h

## Error Handling

errno.h

## Non-local Jumping

setjmp.h

## Runtime Assertions

assert.h

## Process Signaling

signal.h

# String Functions – string.h

---

# Sections

## Manipulation

String alteration

## Comparison

String equality

## Metadata

Retrieve information  
about a string

## Error Handling

Generating error  
strings

## Memory

Memory manipulation  
functions...

# String Manipulation Functions

Function Signature	Purpose
<b>char*</b> strcpy( <b>char*</b> destination, <b>const char*</b> source)	Copies a given source string, including the null terminator to the destination string.
<b>char*</b> strncpy( <b>char*</b> destination, <b>const char*</b> source, <b>size_t</b> n)	Copies “n” characters from the source string to the destination string
<b>char*</b> strcat( <b>char*</b> destination, <b>const char*</b> source)	Appends the source string to the end of the destination string
<b>char*</b> strncat( <b>char*</b> destination, <b>const char*</b> source, <b>size_t</b> n)	Appends “n” characters from the source string on to the end of the destination string
<b>size_t</b> strxfrm( <b>char*</b> destination, <b>const char*</b> source, <b>size_t</b> n)	Transforms up to “n” characters from the source string into the current locale and puts them in the destination string
<b>char*</b> strtok( <b>char*</b> str, <b>const char*</b> delimiter)	Tokenizes the given string using the given delimiter

# String Comparison Functions

Function Signature	Purpose
<b>int strcmp</b> (const char* str1, const char* str2)	Compares two strings for equality and returns zero if the strings are equal
<b>int strncmp</b> (const char* str1, const char* str2, size_t n)	Compares two strings for equality up to a max of “n” characters and returns zero if the strings are equal
<b>int strcoll</b> (const char* str1, const char* str2)	Compares two strings using the locale defined by the LC_COLLATE macro variable and returns zero if the strings are equal

# String Metadata Functions

Function Signature	Purpose
<b>size_t</b> <b>strlen</b> ( <b>const char*</b> str)	Returns the length of the string not including the null terminator character
<b>char*</b> <b>strchr</b> ( <b>const char*</b> str, <b>int</b> character)	Finds the first occurrence of the given character within the given string
<b>char*</b> <b>strrchr</b> ( <b>const char*</b> str, <b>int</b> character)	Finds the last occurrence of the given character within the given string
<b>size_t</b> <b>strspn</b> ( <b>const char*</b> str1, <b>const char*</b> str2)	Returns the count of matching characters from the beginning of str1 that match characters in str2
<b>size_t</b> <b>strcspn</b> ( <b>const char*</b> str1, <b>const char*</b> str2)	Returns the count of non-matching characters from the beginning of str1 when compared with str2
<b>char*</b> <b>strpbrk</b> ( <b>const char*</b> str1, <b>const char*</b> str2)	Finds the first character in str1 that matches any character in str2 not including the null terminator character
<b>char*</b> <b>strstr</b> ( <b>const char*</b> str1, <b>const char*</b> str2)	Finds and returns a pointer to the first occurrence of the substring str2 within str1 not counting null terminator characters

# Error Handling Functions

Function Signature	Purpose
<code>char* strerror(int error_number)</code>	Works with the <code>errno.h</code> header file to print the error string associated with a given library or system error defined by the <code>errno</code> thread-local global variable.



# Memory Functions

Function Signature	Purpose
<b>void*</b> <b>memset</b> ( <b>void*</b> mem, <b>int</b> character, <b>size_t</b> n)	Given a pointer to some memory, a character and a size in bytes, memset fills the memory with the value of the given character up to the given size
<b>void*</b> <b>memcpy</b> ( <b>void*</b> dest, <b>const void*</b> source, <b>size_t</b> n)	Copies “n” bytes in memory from the source to the destination. Assumes that these two chunks of memory do not overlap
<b>void*</b> <b>memmove</b> ( <b>void*</b> dest, <b>const void*</b> source, <b>size_t</b> n)	Does the same thing as memcpy except the two chunks of memory can overlap
<b>int</b> <b>memcmp</b> ( <b>const void*</b> mem1, <b>const void*</b> mem2, <b>size_t</b> n)	Compares up to “n” bytes of two chunks of memory and returns zero if they are equal
<b>void*</b> <b>memchr</b> ( <b>const void*</b> mem, <b>int</b> character, <b>size_t</b> n)	Finds the first occurrence of the given character within the first “n” bytes of the given block of memory

Up Next:

Demo: String Functions

---

# Demo



- **String Functions**
- **Example function usage:**
  - **Manipulation**
  - **Comparison**
  - **Metadata/Info**
  - **Error handling**
  - **Miscellaneous memory**

# Character Testing – ctype.h

---

# Character Testing Functions

Function Signature	Purpose
<code>int isalnum(int c)</code>	Checks whether the character is alphanumeric
<code>int isalpha(int c)</code>	Checks whether the character is alphabetic
<code>int iscntrl(int c)</code>	Checks whether the character is the “control” character
<code>int isdigit(int c)</code>	Checks whether the character is a decimal digit
<code>int isxdigit(int c)</code>	Checks whether the character is a hexadecimal digit
<code>int islower(int c)</code>	Checks whether the character is lowercase
<code>int isupper(int c)</code>	Checks whether the character is uppercase
<code>int ispunct(int c)</code>	Checks whether the character is punctuation

# Character Testing Functions

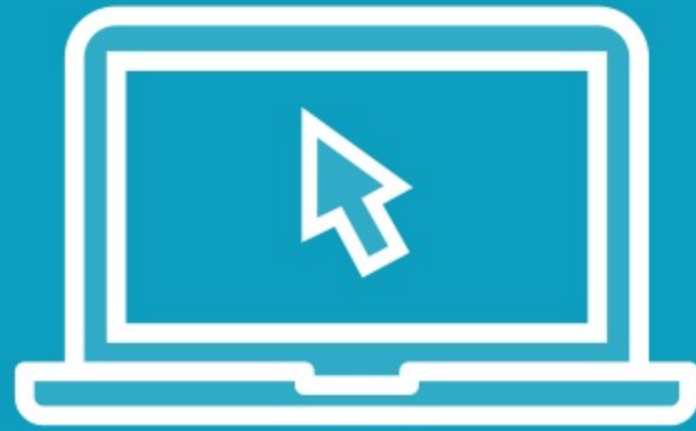
Function Signature	Purpose
<code>int isspace(int c)</code>	Checks whether the character is whitespace
<code>int isprint(int c)</code>	Checks whether the character is printable (any character except the "control" character)
<code>int isgraph(int c)</code>	Checks whether the character can be represented graphically (any printable character except whitespace characters)
<code>int tolower(int c)</code>	Converts a character to lowercase
<code>int toupper(int c)</code>	Converts a character to uppercase

Up Next:

Demo: Character Testing Functions

---

# Demo



- **Character testing functions**
- **Wired Brain Coffee library wrapper function**



# Error Handling Macros – errno.h

---

# Error Handling Macros

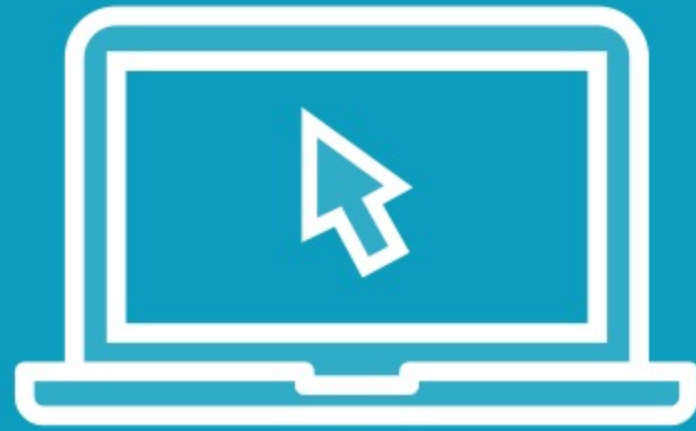
Macro Description	Purpose
<b>extern int errno</b>	This global variable is used by system calls and C standard library functions in order to communicate an error code
<b>int EDOM</b>	This macro can be compared against the current value of "errno" and represents a domain error when using a mathematical function
<b>int ERANGE</b>	This macro can be compared against the current value of "errno" and represents a range error when using a mathematical function

Up Next:

Demo: Error Handling Macros

---

# Demo



- See examples of how to use `errno` with library function errors
  - `EDOM`
  - `ERANGE`

# Non-local Jumping – setjmp.h

---

# Non-local Jumping Functions/Macros

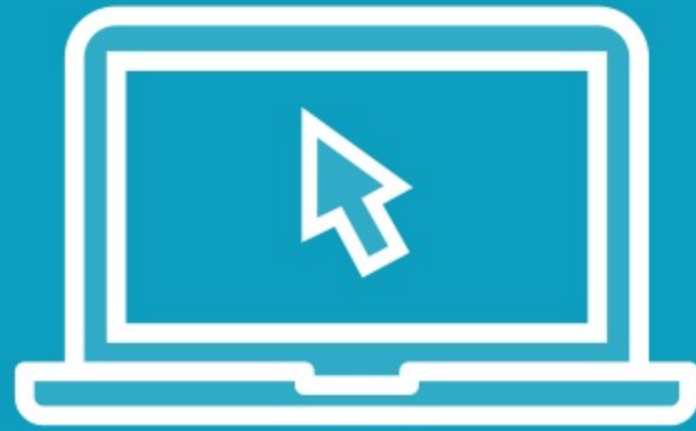
Function/Macro Signature	Purpose
<b>int</b> <b>setjmp</b> ( <b>jmp_buf</b> position)	<p>Calling this macro will save the current code environment (stack pointer, frame pointer, and program counter) given a jmp_buf position variable.</p> <p>It returns zero if called directly from setjmp.</p>
<b>void</b> <b>longjmp</b> ( <b>jmp_buf</b> position, <b>int</b> value)	<p>Resets the stack pointer, frame pointer and program counter such that program execution begins at the point at which you called setjmp with the given “position” jmp_buf variable</p>

Up Next:

Demo: Non-local Jumping

---

# Demo



- **setjmp**
- **longjmp**
- **Discuss try/catch control flow in C**



# Runtime Assertions – assert.h

---

# Runtime Assertion Macro

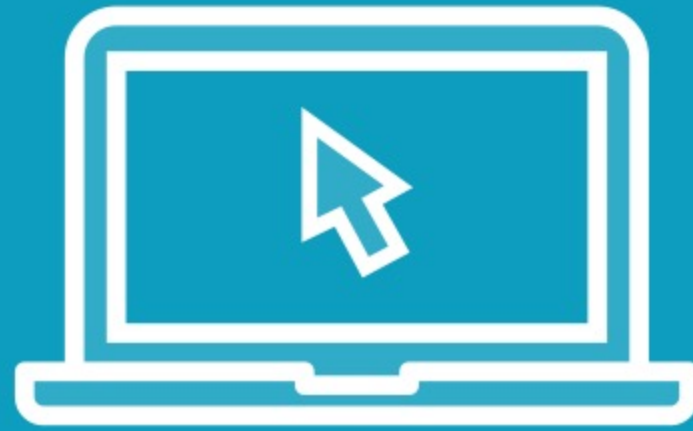
Macro	Purpose
<code>void assert(int expression)</code>	<p>If the expression fed into this macro evaluates to true, nothing happens.</p> <p>If the expression evaluates to false, the macro will write a message to stderr and abort program execution.</p>

Up Next:

Demo: Runtime Assertions

---

# Demo



- **Demonstrate successful assertions**
- **Demonstrate program termination on failed assertions**

# Process Signaling – signal.h

---

# Process Signaling Macros

Macro	Purpose
<b>SIGABRT</b>	Signals abnormal program termination
<b>SIGFPE</b>	Signals that a floating-point error occurred
<b>SIGILL</b>	Signals that an illegal operation occurred
<b>SIGINT</b>	Signals that an interrupt occurred (ctrl-c)
<b>SIGSEGV</b>	Signals violation of memory/data access
<b>SIGTERM</b>	Signals program termination

# Process Signaling Functions

Function Signature	Purpose
<code>void ( *signal( int signal, void ( *foo )( int ) ) )( int )</code>	Given a signal macro (defined on the previous slide) and a pointer to a function, foo, that returns void and takes in an integer, signal will return a pointer to the given function and also register it as a signal handler for the given signal.
<code>int raise(int signal)</code>	<p>This function raises the process signal passed to it.</p> <p>This function works in tandem with the macros described on the previous slide.</p>

# Predefined Signal Handling Functions

Macro	Purpose
<b>SIG_DFL</b>	Default signal handling function used with the “signal” library function.
<b>SIG_IGN</b>	A macro that expands to a predefined function which ignores a given signal.



Up Next:

Demo: Process Signaling

---

# Demo



- Define a signal handler for the SIGABRT signal
- How to raise a signal
- Predefined signal handling function use

# Summary



- **Standard library functions:**
  - **Strings**
  - **Characters**
  - **Error handling**
  - **Non-local jumping and control flow**
  - **Runtime assertions**
  - **Process signaling**