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

String Comparison Functions

Function Signature	Purpose
int strcmp(const char* str1, const char* str2)	Compares two strings for equality and returns zero if the strings are equal
int strncmp(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
int strcoll(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
size_t strlen(const char* str)	Returns the length of the string not including the null terminator character
char* strchr(const char* str, int character)	Finds the first occurrence of the given character within the given string
char* strrchr(const char* str, int character)	Finds the last occurrence of the given character within the given string
size_t strspn(const char* str1, const char* str2)	Returns the count of matching characters from the beginning of str1 that match characters in str2
size_t strcspn(const char* str1, const char* str2)	Returns the count of non-matching characters from the beginning of str1 when compared with str2
char* strpbrk(const char* str1, const char* str2)	Finds the first character in str1 that matches any character in str2 not including the null terminator character
char* strstr(const char* str1, const char* 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
char* strerror(int error_number)	Works with the errno.h header file to print the error string associated with a given library or system error defined by the errno thread-local global variable.

Memory Functions

Function Signature	Purpose
void* memset(void* mem, int character, size_t 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
void* memcpy(void* dest, const void* source, size_t n)	Copies “n” bytes in memory from the source to the destination. Assumes that these two chunks of memory do not overlap
void* memmove(void* dest, const void* source, size_t n)	Does the same thing as memcpy except the two chunks of memory can overlap
int memcmp(const void* mem1, const void* mem2, size_t n)	Compares up to “n” bytes of two chunks of memory and returns zero if they are equal
void* memchr(const void* mem, int character, size_t 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
int isalnum(int c)	Checks whether the character is alphanumeric
int isalpha(int c)	Checks whether the character is alphabetic
int iscntrl(int c)	Checks whether the character is the “control” character
int isdigit(int c)	Checks whether the character is a decimal digit
int isxdigit(int c)	Checks whether the character is a hexadecimal digit
int islower(int c)	Checks whether the character is lowercase
int isupper(int c)	Checks whether the character is uppercase
int ispunct(int c)	Checks whether the character is punctuation

Character Testing Functions

Function Signature	Purpose
int isspace(int c)	Checks whether the character is whitespace
int isprint(int c)	Checks whether the character is printable (any character except the ”control” character)
int isgraph(int c)	Checks whether the character can be represented graphically (any printable character except whitespace characters)
int tolower(int c)	Converts a character to lowercase
int toupper(int c)	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
extern int errno	This global variable is used by system calls and C standard library functions in order to communicate an error code
int EDOM	This macro can be compared against the current value of ”errno” and represents a domain error when using a mathematical function
int ERANGE	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
int setjmp(jmp_buf position)	Calling this macro will save the current code environment (stack pointer, frame pointer, and program counter) given a jmp_buf position variable. It returns zero if called directly from setjmp.
void longjmp(jmp_buf position, int value)	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

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
`void assert(int expression)`	If the expression fed into this macro evaluates to true, nothing happens. If the expression evaluates to false, the macro will write a message to stderr and abort program execution.

Up Next:

Demo: Runtime Assertions

Demo

- Demonstrate successful assertions
- Demonstrate program termination on failed assertions

Process Signaling – signal.h

Process Signaling Macros

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

Process Signaling Functions

Function Signature	Purpose
`void ( *signal( int signal, void ( *foo )( int ) ) )( int )`	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.
`int raise(int signal)`	This function raises the process signal passed to it. This function works in tandem with the macros described on the previous slide.

Predefined Signal Handling Functions

Macro	Purpose
SIG_DFL	Default signal handling function used with the “signal” library function.
SIG_IGN	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