

Almog String Manipulation

Generated by Doxygen 1.9.1

1 File Index	1
1.1 File List	1
2 File Documentation	3
2.1 Almog_String_Manipulation.h File Reference	3
2.1.1 Detailed Description	6
2.1.2 Macro Definition Documentation	6
2.1.2.1 asm_dprintCHAR	6
2.1.2.2 asm_dprintDOUBLE	7
2.1.2.3 asm_dprintFLOAT	7
2.1.2.4 asm_dprintINT	7
2.1.2.5 asm_dprintSIZE_T	8
2.1.2.6 asm_dprintSTRING	8
2.1.2.7 asm_max	8
2.1.2.8 ASM_MAX_LEN	9
2.1.2.9 asm_min	9
2.1.3 Function Documentation	10
2.1.3.1 asm_check_char_belong_to_base()	10
2.1.3.2 asm_copy_array_by_indexes()	10
2.1.3.3 asm_get_char_value_in_base()	11
2.1.3.4 asm_get_line()	12
2.1.3.5 asm_get_next_token_from_str()	12
2.1.3.6 asm_get_token_and_cut()	13
2.1.3.7 asm_isalnum()	14
2.1.3.8 asm_isalpha()	14
2.1.3.9 asm_iscntrl()	15
2.1.3.10 asm_isdigit()	15
2.1.3.11 asm_isgraph()	16
2.1.3.12 asm_islower()	16
2.1.3.13 asm_isprint()	17
2.1.3.14 asm_ispunct()	17
2.1.3.15 asm_isspace()	17
2.1.3.16 asm_isupper()	18
2.1.3.17 asm_isxdigit()	18
2.1.3.18 asm_isXdigit()	19
2.1.3.19 asm_left_pad()	19
2.1.3.20 asm_left_shift()	20
2.1.3.21 asm_length()	20
2.1.3.22 asm_memset()	21
2.1.3.23 asm_print_many_times()	22
2.1.3.24 asm_remove_char_form_string()	22
2.1.3.25 asm_str2double()	23

2.1.3.26 asm_str2float()	23
2.1.3.27 asm_str2int()	24
2.1.3.28 asm_str2size_t()	25
2.1.3.29 asm_str_in_str()	25
2.1.3.30 asm_str_is_whitespace()	26
2.1.3.31 asm_strip_whitespace()	26
2.1.3.32 asm_strncat()	27
2.1.3.33 asm_strncmp()	28
2.1.3.34 asm_tolower()	28
2.1.3.35 asm_toupper()	29
2.2 Almog_String_Manipulation.h	29
2.3 temp.c File Reference	36
2.3.1 Macro Definition Documentation	37
2.3.1.1 ALMOG_STRING_MANIPULATION_IMPLEMENTATION	37
2.3.2 Function Documentation	37
2.3.2.1 main()	37
2.4 temp.c	37
2.5 tests.c File Reference	38
2.5.1 Macro Definition Documentation	39
2.5.1.1 ALMOG_STRING_MANIPULATION_IMPLEMENTATION	39
2.5.1.2 NO_ERRORS	39
2.5.1.3 TEST_CASE	39
2.5.1.4 TEST_EQ_INT	40
2.5.1.5 TEST_EQ_SIZE	40
2.5.1.6 TEST_EQ_STR	40
2.5.1.7 TEST_NE_STR	40
2.5.1.8 TEST_WARN	40
2.5.2 Function Documentation	41
2.5.2.1 fill_sentinel()	41
2.5.2.2 is_nul_terminated_within()	41
2.5.2.3 main()	41
2.5.2.4 rand_ascii_printable()	42
2.5.2.5 test_ascii_classification_exhaustive_ranges()	42
2.5.2.6 test_ascii_classification_full_scan_0_127()	42
2.5.2.7 test_base_digit_helpers()	42
2.5.2.8 test_case_conversion_roundtrip()	43
2.5.2.9 test_copy_array_by_indexes_behavior_and_bounds()	43
2.5.2.10 test_get_line_tmpfile()	43
2.5.2.11 test_get_line_too_long()	43
2.5.2.12 test_get_next_word_from_line_current_behavior()	44
2.5.2.13 test_get_word_and_cut_edges()	44
2.5.2.14 test_left_pad_edges_and_sentinel()	44

2.5.2.15 test_left_shift_edges()	44
2.5.2.16 test_length_matches_strlen_small()	45
2.5.2.17 test_memset_basic_and_edges()	45
2.5.2.18 test_remove_char_from_string_edges()	45
2.5.2.19 test_str2float_double()	45
2.5.2.20 test_str2int()	46
2.5.2.21 test_str2size_t()	46
2.5.2.22 test_str_in_str_overlap_and_edges()	46
2.5.2.23 test_str_is_whitespace_edges()	46
2.5.2.24 test_strip_whitespace_properties()	47
2.5.2.25 test_strncat_current_behavior_and_sentinel()	47
2.5.2.26 test_strncmp_boolean_edges()	47
2.5.2.27 xorshift32()	47
2.5.3 Variable Documentation	48
2.5.3.1 g_tests_failed	48
2.5.3.2 g_tests_run	48
2.5.3.3 g_tests_warned	48
2.5.3.4 rng_state	48
2.6 tests.c	49
Index	59

Chapter 1

File Index

1.1 File List

Here is a list of all files with brief descriptions:

Almog_String_Manipulation.h	
Lightweight string and line manipulation helpers	3
temp.c	36
tests.c	38

Chapter 2

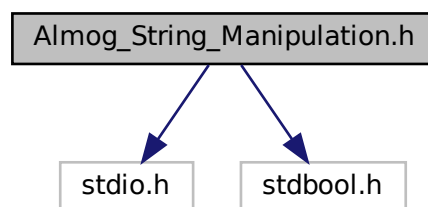
File Documentation

2.1 Almog_String_Manipulation.h File Reference

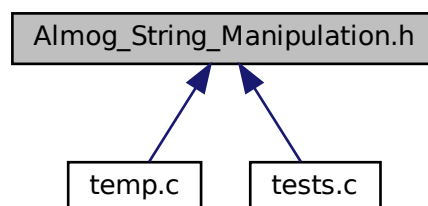
Lightweight string and line manipulation helpers.

```
#include <stdio.h>
#include <stdbool.h>
```

Include dependency graph for Almog_String_Manipulation.h:



This graph shows which files directly or indirectly include this file:



Macros

- `#define ASM_MAX_LEN (int)1e3`
Maximum number of characters processed in some string operations.
- `#define asm_dprintSTRING(expr) printf(#expr " = %s\n", expr)`
Debug-print a C string expression as "expr = value\n".
- `#define asm_dprintCHAR(expr) printf(#expr " = %c\n", expr)`
Debug-print a character expression as "expr = c\n".
- `#define asm_dprintINT(expr) printf(#expr " = %d\n", expr)`
Debug-print an integer expression as "expr = n\n".
- `#define asm_dprintFLOAT(expr) printf(#expr " = %g\n", expr)`
Debug-print a float expression as "expr = n\n".
- `#define asm_dprintDOUBLE(expr) printf(#expr " = %g\n", expr)`
Debug-print a double expression as "expr = n\n".
- `#define asm_dprintSIZE_T(expr) printf(#expr " = %zu\n", expr)`
Debug-print a size_t expression as "expr = n\n".
- `#define asm_min(a, b) ((a) < (b) ? (a) : (b))`
Return the smaller of two values (macro).
- `#define asm_max(a, b) ((a) > (b) ? (a) : (b))`
Return the larger of two values (macro).

Functions

- `bool asm_check_char_belong_to_base (const char c, const size_t base)`
Check if a character is a valid digit in a given base.
- `void asm_copy_array_by_indexes (char *const target, const int start, const int end, const char *const src)`
Copy a substring from src into target by indices and null-terminate.
- `int asm_get_char_value_in_base (const char c, const size_t base)`
Convert a digit character to its numeric value in base-N.
- `int asm_get_line (FILE *fp, char *const dst)`
Read a single line from a stream into a buffer.
- `int asm_get_next_token_from_str (char *const dst, const char *const src, const char delimiter)`
Copy characters from the start of a string into a token buffer.
- `int asm_get_token_and_cut (char *const dst, char *src, const char delimiter, const bool leave_delimiter)`
Get the next word and cut the source string at that point.
- `bool asm_isalnum (char c)`
Test for an alphanumeric character (ASCII).
- `bool asm_isalpha (char c)`
Test for an alphabetic character (ASCII).
- `bool asm_iscntrl (char c)`
Test for a control character (ASCII).
- `bool asm_isdigit (char c)`
Test for a decimal digit (ASCII).
- `bool asm_isgraph (char c)`
Test for any printable character except space (ASCII).
- `bool asm_islower (char c)`
Test for a lowercase letter (ASCII).
- `bool asm_isprint (char c)`
Test for any printable character including space (ASCII).
- `bool asm_ispunct (char c)`

- Test for a punctuation character (ASCII).*

 - bool [asm_isspace](#) (char c)
- Test for a whitespace character (ASCII).*

 - bool [asm_isupper](#) (char c)
- Test for an uppercase letter (ASCII).*

 - bool [asm_isxdigit](#) (char c)
- Test for a hexadecimal digit (lowercase or decimal).*

 - bool [asm_isXdigit](#) (char c)
- Test for a hexadecimal digit (uppercase or decimal).*

 - void [asm_left_pad](#) (char *const s, const size_t padding, const char pad)
- Left-pad a string in-place.*

 - void [asm_left_shift](#) (char *const s, const size_t shift)
- Shift a string left in-place by *shift* characters.*

 - size_t [asm_length](#) (const char *const str)
- Compute the length of a null-terminated C string.*

 - void * [asm_memset](#) (void *const des, const unsigned char value, const size_t n)
- Set a block of memory to a repeated byte value.*

 - void [asm_print_many_times](#) (const char *const str, const size_t n)
- Print a string *n* times, then print a newline.*

 - void [asm_remove_char_from_string](#) (char *const s, const size_t index)
- Remove a single character from a string by index.*

 - int [asm_str_in_str](#) (const char *const src, const char *const word_to_search)
- Count occurrences of a substring within a string.*

 - double [asm_str2double](#) (const char *const s, const char **const end, const size_t base)
- Convert a string to double in the given base.*

 - float [asm_str2float](#) (const char *const s, const char **const end, const size_t base)
- Convert a string to float in the given base.*

 - int [asm_str2int](#) (const char *const s, const char **const end, const size_t base)
- Convert a string to int in the given base.*

 - size_t [asm_str2size_t](#) (const char *const s, const char **const end, const size_t base)
- Convert a string to size_t in the given base.*

 - void [asm_strip_whitespace](#) (char *const s)
- Remove all ASCII whitespace characters from a string in-place.*

 - bool [asm_str_is_whitespace](#) (const char *const s)
- Check whether a string contains only ASCII whitespace characters.*

 - int [asm_strncat](#) (char *const s1, const char *const s2, const int N)
- Append up to *N* characters from *s2* to the end of *s1*.*

 - int [asm_strncmp](#) (const char *s1, const char *s2, const int N)
- Compare up to *N* characters for equality (boolean result).*

 - void [asm_tolower](#) (char *const s)
- Convert all ASCII letters in a string to lowercase in-place.*

 - void [asm_toupper](#) (char *const s)
- Convert all ASCII letters in a string to uppercase in-place.*

2.1.1 Detailed Description

Lightweight string and line manipulation helpers.

This single-header module provides small utilities for working with C strings:

- Reading a single line from a FILE stream
- Measuring string length
- Extracting the next token from a string using a delimiter (skipping leading ASCII whitespace)
- Cutting the extracted token (and leading whitespace) from the source buffer
- Copying a substring by indices
- Counting occurrences of a substring
- A boolean-style strncmp (returns 1 on equality, 0 otherwise)
- ASCII-only character classification helpers (isalnum, isalpha, ...)
- ASCII case conversion (toupper / tolower)
- In-place whitespace stripping and left padding
- Base-N string-to-number conversion for int, size_t, float, and double

Usage

- In exactly one translation unit, define `ALMOG_STRING_MANIPULATION_IMPLEMENTATION` before including this header to compile the implementation.
- In all other files, include the header without the macro to get declarations only.

Notes and limitations

- All destination buffers must be large enough; functions do not grow or allocate buffers.
- `asm_get_line` and `asm_length` enforce `ASM_MAX_LEN` characters (not counting the terminating `'\0'`). Longer lines cause an early return with an error message.
- `asm_strncmp` differs from the standard C `strncmp`: this version returns 1 if equal and 0 otherwise.
- Character classification and case-conversion helpers are ASCII-only and not locale aware.

Definition in file [Almog_String_Manipulation.h](#).

2.1.2 Macro Definition Documentation

2.1.2.1 asm_dprintCHAR

```
#define asm_dprintCHAR(  
    expr ) printf(#expr " = %c\n", expr)
```

Debug-print a character expression as `"expr = c\n"`.

Parameters

<i>expr</i>	An expression that yields a character (or an int promoted from a character). The expression is evaluated exactly once.
-------------	--

Definition at line 82 of file [Almog_String_Manipulation.h](#).

2.1.2.2 asm_dprintDOUBLE

```
#define asm_dprintDOUBLE(  
    expr ) printf(#expr " = %#g\n", expr)
```

Debug-print a double expression as "expr = n\n".

Parameters

<i>expr</i>	An expression that yields a double. The expression is evaluated exactly once.
-------------	---

Definition at line 109 of file [Almog_String_Manipulation.h](#).

2.1.2.3 asm_dprintFLOAT

```
#define asm_dprintFLOAT(  
    expr ) printf(#expr " = %#g\n", expr)
```

Debug-print a float expression as "expr = n\n".

Parameters

<i>expr</i>	An expression that yields a float. The expression is evaluated exactly once.
-------------	--

Definition at line 100 of file [Almog_String_Manipulation.h](#).

2.1.2.4 asm_dprintINT

```
#define asm_dprintINT(  
    expr ) printf(#expr " = %d\n", expr)
```

Debug-print an integer expression as "expr = n\n".

Parameters

<i>expr</i>	An expression that yields an int. The expression is evaluated exactly once.
-------------	---

Definition at line 91 of file [Almog_String_Manipulation.h](#).

2.1.2.5 asm_dprintSIZE_T

```
#define asm_dprintSIZE_T(  
    expr ) printf(#expr " = %zu\n", expr)
```

Debug-print a size_t expression as "expr = n\n".

Parameters

<i>expr</i>	An expression that yields a size_t. The expression is evaluated exactly once.
-------------	---

Definition at line 118 of file [Almog_String_Manipulation.h](#).

2.1.2.6 asm_dprintSTRING

```
#define asm_dprintSTRING(  
    expr ) printf(#expr " = %s\n", expr)
```

Debug-print a C string expression as "expr = value\n".

Parameters

<i>expr</i>	An expression that yields a pointer to char (const or non-const). The expression is evaluated exactly once.
-------------	---

Definition at line 73 of file [Almog_String_Manipulation.h](#).

2.1.2.7 asm_max

```
#define asm_max(  
    a,  
    b ) ((a) > (b) ? (a) : (b))
```

Return the larger of two values (macro).

Parameters

<i>a</i>	First value.
<i>b</i>	Second value.

Returns

The larger of *a* and *b*.

Note

Each parameter is evaluated exactly once.

Definition at line 142 of file [Almog_String_Manipulation.h](#).

2.1.2.8 ASM_MAX_LEN

```
#define ASM_MAX_LEN (int)1e3
```

Maximum number of characters processed in some string operations.

This constant limits:

- The number of characters read by `asm_get_line` from a stream (excluding the terminating null byte).
- The maximum number of characters inspected by `asm_length`.

If `asm_get_line` reads more than `ASM_MAX_LEN` characters before encountering '`'` or EOF, it prints an error to `stderr` and returns -1. In that error case, the contents of the destination buffer are not guaranteed to be null-terminated.

Definition at line 63 of file [Almog_String_Manipulation.h](#).

2.1.2.9 asm_min

```
#define asm_min(  
    a,  
    b ) ((a) < (b) ? (a) : (b))
```

Return the smaller of two values (macro).

Parameters

<i>a</i>	First value.
<i>b</i>	Second value.

Returns

The smaller of *a* and *b*.

Note

Each parameter is evaluated exactly once.

Definition at line 130 of file [Almog_String_Manipulation.h](#).

2.1.3 Function Documentation**2.1.3.1 asm_check_char_belong_to_base()**

```
bool asm_check_char_belong_to_base (
    const char c,
    const size_t base )
```

Check if a character is a valid digit in a given base.

Parameters

<i>c</i>	Character to test (e.g., '0'-'9', 'a'-'z', 'A'-'Z').
<i>base</i>	Numeric base in the range [2, 36].

Returns

true if *c* is a valid digit for *base*, false otherwise.

Note

If *base* is outside [2, 36], an error is printed to stderr and false is returned.

Definition at line 195 of file [Almog_String_Manipulation.h](#).

References [asm_isdigit\(\)](#).

Referenced by [asm_get_char_value_in_base\(\)](#), [asm_str2double\(\)](#), [asm_str2float\(\)](#), [asm_str2int\(\)](#), [asm_str2size_t\(\)](#), and [test_base_digit_helpers\(\)](#).

2.1.3.2 asm_copy_array_by_indexes()

```
void asm_copy_array_by_indexes (
    char *const target,
    const int start,
    const int end,
    const char *const src )
```

Copy a substring from *src* into *target* by indices and null-terminate.

Copies characters with indices *i* = *start*, *start* + 1, ..., *end* from *src* into *target* (note: *end* is inclusive in this implementation), then ensures *target* is null-terminated.

Parameters

<i>target</i>	Destination buffer. Must be large enough to hold (end - start + 1) characters plus the null terminator.
<i>start</i>	Inclusive start index within <code>src</code> (0-based).
<i>end</i>	Inclusive end index within <code>src</code> (must satisfy <code>end >= start</code>).
<i>src</i>	Source string buffer.

Warning

No bounds checking is performed. The caller must ensure valid indices and sufficient target capacity.

Definition at line 230 of file [Almog_String_Manipulation.h](#).

Referenced by [main\(\)](#), and [test_copy_array_by_indexes_behavior_and_bounds\(\)](#).

2.1.3.3 asm_get_char_value_in_base()

```
int asm_get_char_value_in_base (  
    const char c,  
    const size_t base )
```

Convert a digit character to its numeric value in base-N.

Parameters

<i>c</i>	Digit character ('0'-'9', 'a'-'z', 'A'-'Z').
<i>base</i>	Numeric base in the range [2, 36] (used for validation).

Returns

The numeric value of `c` in the range [0, 35].

Note

This function assumes `c` is a valid digit character. Call [asm_check_char_belong_to_base\(\)](#) first if validation is needed.

Definition at line 253 of file [Almog_String_Manipulation.h](#).

References [asm_check_char_belong_to_base\(\)](#), [asm_isdigit\(\)](#), and [asm_isupper\(\)](#).

Referenced by [asm_str2double\(\)](#), [asm_str2float\(\)](#), [asm_str2int\(\)](#), [asm_str2size_t\(\)](#), and [test_base_digit_helpers\(\)](#).

2.1.3.4 `asm_get_line()`

```
int asm_get_line (
    FILE * fp,
    char *const dst )
```

Read a single line from a stream into a buffer.

Reads characters from the FILE stream until a newline ('
') or EOF is encountered. The newline, if present, is not copied. The result is always null-terminated on normal (non-error) completion.

Parameters

<i>fp</i>	Input stream (must be non-NULL).
<i>dst</i>	Destination buffer. Must have capacity of at least ASM_MAX_LEN + 1 bytes.

Returns

Number of characters stored in *dst* (excluding the terminating null byte).

Return values

-1	EOF was encountered before any character was read, or the line exceeded ASM_MAX_LEN characters (error).
----	---

Note

If the line exceeds ASM_MAX_LEN characters before a newline or EOF is seen, the function prints an error message to stderr and returns -1. In that case, *dst* is not guaranteed to be null-terminated.

An empty line (just '
') returns 0 (not -1).

Definition at line 285 of file [Almog_String_Manipulation.h](#).

References [ASM_MAX_LEN](#).

Referenced by [test_get_line_tmpfile\(\)](#), and [test_get_line_too_long\(\)](#).

2.1.3.5 `asm_get_next_token_from_str()`

```
int asm_get_next_token_from_str (
    char *const dst,
    const char *const src,
    const char delimiter )
```

Copy characters from the start of a string into a token buffer.

Copies characters from *src* into *dst* until one of the following is encountered in *src*:

- the delimiter character,
- or the string terminator ('\0').

The delimiter (if present) is not copied into *dst*. The resulting token in *dst* is always null-terminated.

Parameters

<i>dst</i>	Destination buffer for the extracted token. Must be large enough to hold the token plus the null terminator.
<i>src</i>	Source C string to parse (not modified by this function).
<i>delimiter</i>	Delimiter character to stop at.

Returns

The number of characters copied into `dst` (excluding the null terminator). This is also the index in `src` of the delimiter or `'\0'` that stopped the copy.

Note

This function does not skip leading whitespace and does not treat newline (`'\n'`) specially; newlines are copied like any other character.

If `src` starts with `delimiter` or `'\0'`, an empty token is produced (`dst` becomes `""`), and 0 is returned.

Definition at line 331 of file [Almog_String_Manipulation.h](#).

Referenced by [asm_get_token_and_cut\(\)](#), and [test_get_next_word_from_line_current_behavior\(\)](#).

2.1.3.6 asm_get_token_and_cut()

```
int asm_get_token_and_cut (
    char *const dst,
    char * src,
    const char delimiter,
    const bool leave_delimiter )
```

Get the next word and cut the source string at that point.

Extracts the next word from `src` (per `asm_get_next_word_from_line` semantics) into `dst`. On success, `src` is modified in-place to remove the consumed prefix.

If `leave_delimiter` is true, the new `src` begins at the delimiter character. If false, the delimiter is skipped and the new `src` begins right after it.

Example (`leave_delimiter == true`):

```
char src[] = "abc,def";
char word[4];
asm_get_word_and_cut(word, src, ',', true);
// word == "abc"
// src == ",def"
```

Parameters

<i>dst</i>	Destination buffer for the extracted word (large enough for the token and terminating null).
<i>src</i>	Source buffer. Modified in-place if a word is found.
<i>delimiter</i>	Delimiter character to stop at.
<i>leave_delimiter</i>	If true, the delimiter remains at the start of the updated <code>src</code> ; if false, it is removed as well.

Returns

1 if a non-empty token was extracted into `dst`, 0 otherwise.

Note

Even when this function returns 0, it may still modify `src` if:

- leading whitespace was consumed, and/or
- `leave_delimiter` is false and the delimiter was the first non-whitespace character.

Definition at line 377 of file [Almog_String_Manipulation.h](#).

References [asm_get_next_token_from_str\(\)](#), and [asm_left_shift\(\)](#).

Referenced by [main\(\)](#), and [test_get_word_and_cut_edges\(\)](#).

2.1.3.7 asm_isalnum()

```
bool asm_isalnum (
    char c )
```

Test for an alphanumeric character (ASCII).

Parameters

<code>c</code>	Character to test.
----------------	--------------------

Returns

true if `c` is '0'-'9', 'A'-'Z', or 'a'-'z'; false otherwise.

Definition at line 395 of file [Almog_String_Manipulation.h](#).

References [asm_isalpha\(\)](#), and [asm_isdigit\(\)](#).

Referenced by [test_ascii_classification_exhaustive_ranges\(\)](#), and [test_ascii_classification_full_scan_0_127\(\)](#).

2.1.3.8 asm_isalpha()

```
bool asm_isalpha (
    char c )
```

Test for an alphabetic character (ASCII).

Parameters

<code>c</code>	Character to test.
----------------	--------------------

Returns

true if `c` is 'A'-'Z' or 'a'-'z'; false otherwise.

Definition at line 406 of file [Almog_String_Manipulation.h](#).

References [asm_islower\(\)](#), and [asm_isupper\(\)](#).

Referenced by [asm_isalnum\(\)](#), [test_ascii_classification_exhaustive_ranges\(\)](#), and [test_ascii_classification_full_scan_0_127\(\)](#).

2.1.3.9 asm_iscntrl()

```
bool asm_iscntrl (  
    char c )
```

Test for a control character (ASCII).

Parameters

<code>c</code>	Character to test.
----------------	--------------------

Returns

true if `c` is in the range [0, 31] or 127; false otherwise.

Definition at line 417 of file [Almog_String_Manipulation.h](#).

Referenced by [test_ascii_classification_exhaustive_ranges\(\)](#).

2.1.3.10 asm_isdigit()

```
bool asm_isdigit (  
    char c )
```

Test for a decimal digit (ASCII).

Parameters

<code>c</code>	Character to test.
----------------	--------------------

Returns

true if `c` is '0'–'9'; false otherwise.

Definition at line 432 of file [Almog_String_Manipulation.h](#).

Referenced by [asm_check_char_belong_to_base\(\)](#), [asm_get_char_value_in_base\(\)](#), [asm_isalnum\(\)](#), [asm_isxdigit\(\)](#), [asm_isXdigit\(\)](#), [test_ascii_classification_exhaustive_ranges\(\)](#), and [test_ascii_classification_full_scan_0_127\(\)](#).

2.1.3.11 asm_isgraph()

```
bool asm_isgraph (  
    char c )
```

Test for any printable character except space (ASCII).

Parameters

<code>c</code>	Character to test.
----------------	--------------------

Returns

true if `c` is in the range [33, 126]; false otherwise.

Definition at line 447 of file [Almog_String_Manipulation.h](#).

Referenced by [asm_isprint\(\)](#), [test_ascii_classification_exhaustive_ranges\(\)](#), and [test_ascii_classification_full_scan_0_127\(\)](#).

2.1.3.12 asm_islower()

```
bool asm_islower (  
    char c )
```

Test for a lowercase letter (ASCII).

Parameters

<code>c</code>	Character to test.
----------------	--------------------

Returns

true if `c` is 'a'–'z'; false otherwise.

Definition at line 462 of file [Almog_String_Manipulation.h](#).

Referenced by [asm_isalpha\(\)](#), [asm_toupper\(\)](#), [test_ascii_classification_exhaustive_ranges\(\)](#), and [test_ascii_classification_full_scan_0_127\(\)](#).

2.1.3.13 asm_isprint()

```
bool asm_isprint (
    char c )
```

Test for any printable character including space (ASCII).

Parameters

<i>c</i>	Character to test.
----------	--------------------

Returns

true if *c* is space (' ') or `asm_isgraph(c)` is true; false otherwise.

Definition at line 478 of file [Almog_String_Manipulation.h](#).

References [asm_isgraph\(\)](#).

Referenced by [test_ascii_classification_exhaustive_ranges\(\)](#), and [test_ascii_classification_full_scan_0_127\(\)](#).

2.1.3.14 asm_ispunct()

```
bool asm_ispunct (
    char c )
```

Test for a punctuation character (ASCII).

Parameters

<i>c</i>	Character to test.
----------	--------------------

Returns

true if *c* is a printable, non-alphanumeric, non-space character; false otherwise.

Definition at line 490 of file [Almog_String_Manipulation.h](#).

Referenced by [test_ascii_classification_exhaustive_ranges\(\)](#).

2.1.3.15 asm_isspace()

```
bool asm_isspace (
    char c )
```

Test for a whitespace character (ASCII).

Parameters

<code>c</code>	Character to test.
----------------	--------------------

Returns

true if `c` is one of ' ', '
, '\t', '\v', '\f', or '\r'; false otherwise.

Definition at line 506 of file [Almog_String_Manipulation.h](#).

Referenced by [asm_str2double\(\)](#), [asm_str2float\(\)](#), [asm_str2int\(\)](#), [asm_str2size_t\(\)](#), [asm_str_is_whitespace\(\)](#), [asm_strip_whitespace\(\)](#), [test_ascii_classification_exhaustive_ranges\(\)](#), and [test_strip_whitespace_properties\(\)](#).

2.1.3.16 asm_isupper()

```
bool asm_isupper (  
    char c )
```

Test for an uppercase letter (ASCII).

Parameters

<code>c</code>	Character to test.
----------------	--------------------

Returns

true if `c` is 'A'-'Z'; false otherwise.

Definition at line 522 of file [Almog_String_Manipulation.h](#).

Referenced by [asm_get_char_value_in_base\(\)](#), [asm_isalpha\(\)](#), [asm_tolower\(\)](#), [test_ascii_classification_exhaustive_ranges\(\)](#), and [test_ascii_classification_full_scan_0_127\(\)](#).

2.1.3.17 asm_isxdigit()

```
bool asm_isxdigit (  
    char c )
```

Test for a hexadecimal digit (lowercase or decimal).

Parameters

<code>c</code>	Character to test.
----------------	--------------------

Returns

true if `c` is '0'–'9' or 'a'–'f'; false otherwise.

Definition at line 537 of file [Almog_String_Manipulation.h](#).

References [asm_isdigit\(\)](#).

Referenced by [test_ascii_classification_exhaustive_ranges\(\)](#).

2.1.3.18 asm_isXdigit()

```
bool asm_isXdigit (
    char c )
```

Test for a hexadecimal digit (uppercase or decimal).

Parameters

<code>c</code>	Character to test.
----------------	--------------------

Returns

true if `c` is '0'–'9' or 'A'–'F'; false otherwise.

Definition at line 552 of file [Almog_String_Manipulation.h](#).

References [asm_isdigit\(\)](#).

Referenced by [test_ascii_classification_exhaustive_ranges\(\)](#).

2.1.3.19 asm_left_pad()

```
void asm_left_pad (
    char *const s,
    const size_t padding,
    const char pad )
```

Left-pad a string in-place.

Shifts the contents of `s` to the right by `padding` positions and fills the vacated leading positions with `pad`.

Parameters

<code>s</code>	String to pad. Modified in-place.
<code>padding</code>	Number of leading spaces to insert.
<code>pad</code>	The padding character to insert.

Warning

The buffer backing `s` must have enough capacity for the original string length plus padding and the terminating null byte. No bounds checking is performed.

Definition at line 575 of file [Almog_String_Manipulation.h](#).

References [asm_length\(\)](#).

Referenced by [main\(\)](#), and [test_left_pad_edges_and_sentinel\(\)](#).

2.1.3.20 asm_left_shift()

```
void asm_left_shift (
    char *const s,
    const size_t shift )
```

Shift a string left in-place by `shift` characters.

Removes the first `shift` characters from `s` by moving the remaining characters to the front. The resulting string is always null-terminated.

Parameters

<code>s</code>	String to modify in-place. Must be null-terminated.
<code>shift</code>	Number of characters to remove from the front.

Note

If `shift` is 0, `s` is unchanged.

If `shift` is greater than or equal to the string length, `s` becomes the empty string.

Definition at line 599 of file [Almog_String_Manipulation.h](#).

References [asm_length\(\)](#).

Referenced by [asm_get_token_and_cut\(\)](#), and [test_left_shift_edges\(\)](#).

2.1.3.21 asm_length()

```
size_t asm_length (
    const char *const str )
```

Compute the length of a null-terminated C string.

Parameters

<i>str</i>	Null-terminated string (must be non-NULL).
------------	--

Returns

The number of characters before the terminating null byte.

Note

If more than `ASM_MAX_LEN` characters are scanned without encountering a null terminator, an error is printed to `stderr` and **SIZE_MAX** is returned.

Definition at line 626 of file [Almog_String_Manipulation.h](#).

References [ASM_MAX_LEN](#).

Referenced by [asm_left_pad\(\)](#), [asm_left_shift\(\)](#), [asm_remove_char_form_string\(\)](#), [asm_str_in_str\(\)](#), [asm_str_is_whitespace\(\)](#), [asm_strip_whitespace\(\)](#), [asm_strncat\(\)](#), [asm_tolower\(\)](#), [asm_toupper\(\)](#), [main\(\)](#), and [test_length_matches_strlen_small\(\)](#).

2.1.3.22 asm_memset()

```
void * asm_memset (
    void *const des,
    const unsigned char value,
    const size_t n )
```

Set a block of memory to a repeated byte value.

Writes `value` into each of the first `n` bytes of the memory region pointed to by `des`. This function mirrors the behavior of the standard C `memset()`, but implements it using a simple byte-wise loop.

Parameters

<i>des</i>	Destination memory block to modify. Must point to a valid buffer of at least <code>n</code> bytes.
<i>value</i>	Unsigned byte value to store repeatedly.
<i>n</i>	Number of bytes to set.

Returns

The original pointer `des`.

Note

This implementation performs no optimizations (such as word-sized writes); the memory block is filled one byte at a time.

Behavior is undefined if `des` overlaps with invalid or non-writable memory.

Definition at line 661 of file [Almog_String_Manipulation.h](#).

Referenced by [test_memset_basic_and_edges\(\)](#).

2.1.3.23 `asm_print_many_times()`

```
void asm_print_many_times (
    const char *const str,
    const size_t n )
```

Print a string *n* times, then print a newline.

Parameters

<i>str</i>	String to print (as-is with <code>printf("%s", ...)</code>).
<i>n</i>	Number of times to print <i>str</i> .

Definition at line 676 of file [Almog_String_Manipulation.h](#).

2.1.3.24 `asm_remove_char_form_string()`

```
void asm_remove_char_form_string (
    char *const s,
    const size_t index )
```

Remove a single character from a string by index.

Deletes the character at position *index* from *s* by shifting subsequent characters one position to the left.

Parameters

<i>s</i>	String to modify in-place. Must be null-terminated.
<i>index</i>	Zero-based index of the character to remove.

Note

If *index* is out of range, an error is printed to `stderr` and the string is left unchanged.

Definition at line 696 of file [Almog_String_Manipulation.h](#).

References [asm_length\(\)](#).

Referenced by [asm_strip_whitespace\(\)](#), and [test_remove_char_form_string_edges\(\)](#).

2.1.3.25 asm_str2double()

```
double asm_str2double (
    const char *const s,
    const char **const end,
    const size_t base )
```

Convert a string to double in the given base.

Parses an optional sign, then a sequence of base-N digits, and optionally a fractional part separated by a '.' character.

Parameters

<i>s</i>	String to convert. Leading ASCII whitespace is skipped.
<i>end</i>	If non-NULL, *end is set to point to the first character not used in the conversion.
<i>base</i>	Numeric base in the range [2, 36].

Returns

The converted double value. Returns 0.0 on invalid base.

Note

Only digits '0'-'9', 'a'-'z', and 'A'-'Z' are recognized as base-N digits. No exponent notation (e.g., 'e' or 'p') is supported.

On invalid base, an error is printed to stderr, *end (if non-NULL) is set to *s*, and 0.0 is returned.

Definition at line 756 of file [Almog_String_Manipulation.h](#).

References [asm_check_char_belong_to_base\(\)](#), [asm_get_char_value_in_base\(\)](#), and [asm_isspace\(\)](#).

Referenced by [test_str2float_double\(\)](#).

2.1.3.26 asm_str2float()

```
float asm_str2float (
    const char *const s,
    const char **const end,
    const size_t base )
```

Convert a string to float in the given base.

Identical to `asm_str2double` semantically, but returns a float and uses float arithmetic for the fractional part.

Parameters

<i>s</i>	String to convert. Leading ASCII whitespace is skipped.
<i>end</i>	If non-NULL, *end is set to point to the first character not used in the conversion.
<i>base</i>	Numeric base in the range [2, 36].

Returns

The converted float value. Returns 0.0f on invalid base.

Note

Only digits '0'–'9', 'a'–'z', and 'A'–'Z' are recognized as base-N digits. No exponent notation (e.g., 'e' or 'p') is supported.

On invalid base, an error is printed to stderr, *end (if non-NULL) is set to s, and 0.0f is returned.

Definition at line 818 of file [Almog_String_Manipulation.h](#).

References [asm_check_char_belong_to_base\(\)](#), [asm_get_char_value_in_base\(\)](#), and [asm_isspace\(\)](#).

Referenced by [test_str2float_double\(\)](#).

2.1.3.27 asm_str2int()

```
int asm_str2int (
    const char *const s,
    const char **const end,
    const size_t base )
```

Convert a string to int in the given base.

Parses an optional sign and then a sequence of base-N digits.

Parameters

<i>s</i>	String to convert. Leading ASCII whitespace is skipped.
<i>end</i>	If non-NULL, *end is set to point to the first character not used in the conversion.
<i>base</i>	Numeric base in the range [2, 36].

Returns

The converted int value. Returns 0 on invalid base.

Note

Only digits '0'–'9', 'a'–'z', and 'A'–'Z' are recognized as base-N digits.

On invalid base, an error is printed to stderr, *end (if non-NULL) is set to s, and 0 is returned.

Definition at line 877 of file [Almog_String_Manipulation.h](#).

References [asm_check_char_belong_to_base\(\)](#), [asm_get_char_value_in_base\(\)](#), and [asm_isspace\(\)](#).

Referenced by [test_str2int\(\)](#).

2.1.3.28 asm_str2size_t()

```
size_t asm_str2size_t (
    const char *const s,
    const char **const end,
    const size_t base )
```

Convert a string to `size_t` in the given base.

Parses an optional leading '+' sign, then a sequence of base-N digits. Negative numbers are rejected.

Parameters

<i>s</i>	String to convert. Leading ASCII whitespace is skipped.
<i>end</i>	If non-NULL, *end is set to point to the first character not used in the conversion.
<i>base</i>	Numeric base in the range [2, 36].

Returns

The converted `size_t` value. Returns 0 on invalid base or if a negative sign is encountered.

Note

On invalid base or a negative sign, an error is printed to `stderr`, *end (if non-NULL) is set to *s*, and 0 is returned.

Definition at line 922 of file [Almog_String_Manipulation.h](#).

References [asm_check_char_belong_to_base\(\)](#), [asm_get_char_value_in_base\(\)](#), and [asm_isspace\(\)](#).

Referenced by [test_str2size_t\(\)](#).

2.1.3.29 asm_str_in_str()

```
int asm_str_in_str (
    const char *const src,
    const char *const word_to_search )
```

Count occurrences of a substring within a string.

Counts how many times `word_to_search` appears in `src`. Occurrences may overlap.

Parameters

<i>src</i>	The string to search in (must be null-terminated).
<i>word_to_search</i>	The substring to find (must be null-terminated and non-empty).

Returns

The number of (possibly overlapping) occurrences found.

Note

If `word_to_search` is the empty string, the behavior is not well-defined and should be avoided.

Definition at line 726 of file [Almog_String_Manipulation.h](#).

References [asm_length\(\)](#), and [asm_strncmp\(\)](#).

Referenced by [test_str_in_str_overlap_and_edges\(\)](#).

2.1.3.30 asm_str_is_whitespace()

```
bool asm_str_is_whitespace (
    const char *const s )
```

Check whether a string contains only ASCII whitespace characters.

Parameters

<code>s</code>	Null-terminated string to test.
----------------	---------------------------------

Returns

true if every character in `s` satisfies [asm_isspace\(\)](#), or if `s` is the empty string; false otherwise.

Definition at line 990 of file [Almog_String_Manipulation.h](#).

References [asm_isspace\(\)](#), and [asm_length\(\)](#).

Referenced by [test_str_is_whitespace_edges\(\)](#).

2.1.3.31 asm_strip_whitespace()

```
void asm_strip_whitespace (
    char *const s )
```

Remove all ASCII whitespace characters from a string in-place.

Scans `s` and deletes all characters for which [asm_isspace\(\)](#) is true, compacting the string and preserving the original order of non-whitespace characters.

Parameters

<code>s</code>	String to modify in-place. Must be null-terminated.
----------------	---

Definition at line 969 of file [Almog_String_Manipulation.h](#).

References [asm_isspace\(\)](#), [asm_length\(\)](#), and [asm_remove_char_form_string\(\)](#).

Referenced by [test_strip_whitespace_properties\(\)](#).

2.1.3.32 `asm_strncat()`

```
int asm_strncat (  
    char *const s1,  
    const char *const s2,  
    const int N )
```

Append up to `N` characters from `s2` to the end of `s1`.

Appends characters from `s2` to the end of `s1` until either:

- `N` characters were appended, or
- a `'\0'` is encountered in `s2`.

After appending, this implementation writes a terminating `'\0'` to `s1`.

Parameters

<code>s1</code>	Destination string buffer (must be null-terminated).
<code>s2</code>	Source string buffer (must be null-terminated).
<code>N</code>	Maximum number of characters to append. If <code>N == 0</code> , the limit defaults to <code>ASM_MAX_LEN</code> .

Returns

The number of characters appended to `s1`.

Warning

This function uses `ASM_MAX_LEN` as an upper bound for the resulting length (excluding the terminating `'\0'`). The caller must ensure `s1` has capacity of at least `ASM_MAX_LEN` bytes.

Definition at line 1021 of file [Almog_String_Manipulation.h](#).

References [asm_length\(\)](#), and [ASM_MAX_LEN](#).

Referenced by [test_strncat_current_behavior_and_sentinel\(\)](#).

2.1.3.33 `asm_strncmp()`

```
int asm_strncmp (
    const char * s1,
    const char * s2,
    const int N )
```

Compare up to N characters for equality (boolean result).

Returns 1 if the first N characters of `s1` and `s2` are all equal; otherwise returns 0. Unlike the standard C `strncmp`, which returns 0 on equality and a non-zero value on inequality/order, this function returns a boolean-like result (1 == equal, 0 == different).

Parameters

<code>s1</code>	First string (may be shorter than N).
<code>s2</code>	Second string (may be shorter than N).
<code>N</code>	Number of characters to compare.

Returns

1 if equal for the first N characters, 0 otherwise.

Note

If either string ends before N characters and the other does not, the strings are considered different.

Definition at line 1061 of file [Almog_String_Manipulation.h](#).

Referenced by [asm_str_in_str\(\)](#), and [test_strncmp_boolean_edges\(\)](#).

2.1.3.34 `asm_tolower()`

```
void asm_tolower (
    char *const s )
```

Convert all ASCII letters in a string to lowercase in-place.

Parameters

<code>s</code>	String to modify in-place. Must be null-terminated.
----------------	---

Definition at line 1081 of file [Almog_String_Manipulation.h](#).

References [asm_isupper\(\)](#), and [asm_length\(\)](#).

Referenced by [test_case_conversion_roundtrip\(\)](#).

2.1.3.35 asm_toupper()

```
void asm_toupper (
    char *const s )
```

Convert all ASCII letters in a string to uppercase in-place.

Parameters

s	String to modify in-place. Must be null-terminated.
---	---

Definition at line 1096 of file [Almog_String_Manipulation.h](#).

References [asm_islower\(\)](#), and [asm_length\(\)](#).

Referenced by [test_case_conversion_roundtrip\(\)](#).

2.2 Almog_String_Manipulation.h

```
00001
00041 #ifndef ALMOG_STRING_MANIPULATION_H_
00042 #define ALMOG_STRING_MANIPULATION_H_
00043
00044 #include <stdio.h>
00045 #include <stdbool.h>
00046
00062 #ifndef ASM_MAX_LEN
00063 #define ASM_MAX_LEN (int)1e3
00064 #endif
00065
00073 #define asm_dprintSTRING(expr) printf(#expr " = %s\n", expr)
00074
00082 #define asm_dprintCHAR(expr) printf(#expr " = %c\n", expr)
00083
00091 #define asm_dprintINT(expr) printf(#expr " = %d\n", expr)
00092
00100 #define asm_dprintFLOAT(expr) printf(#expr " = %g\n", expr)
00101
00109 #define asm_dprintDOUBLE(expr) printf(#expr " = %g\n", expr)
00110
00118 #define asm_dprintSIZE_T(expr) printf(#expr " = %zu\n", expr)
00119
00130 #define asm_min(a, b) ((a) < (b) ? (a) : (b))
00131
00142 #define asm_max(a, b) ((a) > (b) ? (a) : (b))
00143
00144 bool    asm_check_char_belong_to_base(const char c, const size_t base);
00145 void    asm_copy_array_by_indexes(char * const target, const int start, const int end, const char *
const src);
00146 int     asm_get_char_value_in_base(const char c, const size_t base);
00147 int     asm_get_line(FILE *fp, char * const dst);
00148 int     asm_get_next_token_from_str(char * const dst, const char * const src, const char delimiter);
00149 int     asm_get_token_and_cut(char * const dst, char *src, const char delimiter, const bool
leave_delimiter);
00150 bool    asm_isalnum(const char c);
00151 bool    asm_isalpha(const char c);
00152 bool    asm_iscntrl(const char c);
00153 bool    asm_isdigit(const char c);
00154 bool    asm_isgraph(const char c);
00155 bool    asm_islower(const char c);
00156 bool    asm_isprint(const char c);
00157 bool    asm ispunct(const char c);
00158 bool    asm_isspace(const char c);
00159 bool    asm_isupper(const char c);
00160 bool    asm_isxdigit(const char c);
00161 bool    asm_isXdigit(const char c);
00162 void    asm_left_pad(char * const s, const size_t padding, const char pad);
00163 void    asm_left_shift(char * const s, const size_t shift);
00164 size_t   asm_length(const char * const str);
00165 void *   asm_memset(void * const des, const unsigned char value, const size_t n);
00166 void     asm_print_many_times(const char * const str, const size_t n);
00167 void     asm_remove_char_form_string(char * const s, const size_t index);
00168 int      asm_str_in_str(const char * const src, const char * const word_to_search);
```

```

00169 double asm_str2double(const char * const s, const char ** const end, const size_t base);
00170 float asm_str2float(const char * const s, const char ** const end, const size_t base);
00171 int asm_str2int(const char * const s, const char ** const end, const size_t base);
00172 size_t asm_str2size_t(const char * const s, const char ** const end, const size_t base);
00173 void asm_strip_whitespace(char * const s);
00174 bool asm_str_is_whitespace(const char * const s);
00175 int asm_strncat(char * const s1, const char * const s2, const int N);
00176 int asm_strncmp(const char * const s1, const char * const s2, const int N);
00177 void asm_tolower(char * const s);
00178 void asm_toupper(char * const s);
00179
00180 #endif /*ALMOG_STRING_MANIPULATION_H_*/
00181
00182 #ifdef ALMOG_STRING_MANIPULATION_IMPLEMENTATION
00183 #undef ALMOG_STRING_MANIPULATION_IMPLEMENTATION
00184
00195 bool asm_check_char_belong_to_base(const char c, const size_t base)
00196 {
00197     if (base > 36 || base < 2) {
00198         #ifndef NO_ERRORS
00199             fprintf(stderr, "%s:%d:\n%s:\n[Error] Supported bases are [2...36]. Inputted: %zu\n\n",
00200                 __FILE__, __LINE__, __func__, base);
00201             #endif
00202             return false;
00203         }
00204         if (base <= 10) {
00205             return c >= '0' && c <= '9'+(char)base-10;
00206         }
00207         if (base > 10) {
00208             return asm_isdigit(c) || (c >= 'A' && c <= ('A'+(char)base-11)) || (c >= 'a' && c <=
00209                 ('a'+(char)base-11));
00210         }
00211         return false;
00212     }
00230 void asm_copy_array_by_indexes(char * const target, const int start, const int end, const char * const
00231     src)
00232     {
00233         if (start > end) return;
00234         int j = 0;
00235         for (int i = start; i <= end; i++) {
00236             target[j] = src[i];
00237             j++;
00238         }
00239         if (target[j-1] != '\0') {
00240             target[j] = '\0';
00241         }
00242     }
00253 int asm_get_char_value_in_base(const char c, const size_t base)
00254 {
00255     if (!asm_check_char_belong_to_base(c, base)) return -1;
00256     if (asm_isdigit(c)) {
00257         return c - '0';
00258     } else if (asm_isupper(c)) {
00259         return c - 'A' + 10;
00260     } else {
00261         return c - 'a' + 10;
00262     }
00263 }
00264
00285 int asm_get_line(FILE *fp, char * const dst)
00286 {
00287     int i = 0;
00288     int c;
00289     while ((c = fgetc(fp)) != '\n' && c != EOF) {
00290         dst[i++] = c;
00291         if (i >= ASM_MAX_LEN) {
00292             #ifndef NO_ERRORS
00293                 fprintf(stderr, "%s:%d:\nIn function '%s':\n[Error] index exceeds ASM_MAX_LEN. Line in
00294                     file is too long.\n\n", __FILE__, __LINE__, __func__);
00295             #endif
00296             return -1;
00297         }
00298     }
00299     dst[i] = '\0';
00300     if (c == EOF && i == 0) {
00301         return -1;
00302     }
00303     return i;
00304 }
00331 int asm_get_next_token_from_str(char * const dst, const char * const src, const char delimiter)
00332 {
00333     int i = 0, j = 0;
00334     char c;

```

```

00335     while ((c = src[i]) != delimiter && c != '\0') {
00336         dst[j++] = src[i++];
00337     }
00338     dst[j] = '\0';
00339
00340     return j;
00341 }
00342
00343
00377 int asm_get_token_and_cut(char * const dst, char *src, const char delimiter, const bool
    leave_delimiter)
00378 {
00379     int new_src_start_index = asm_get_next_token_from_str(dst, src, delimiter);
00380
00381     if (leave_delimiter) {
00382         asm_left_shift(src, new_src_start_index);
00383     } else {
00384         asm_left_shift(src, new_src_start_index + 1);
00385     }
00386     return new_src_start_index ? 1 : 0;
00387 }
00388
00395 bool asm_isalnum(char c)
00396 {
00397     return asm_isalpha(c) || asm_isdigit(c);
00398 }
00399
00406 bool asm_isalpha(char c)
00407 {
00408     return asm_isupper(c) || asm_islower(c);
00409 }
00410
00417 bool asm_iscntrl(char c)
00418 {
00419     if ((c >= 0 && c <= 31) || c == 127) {
00420         return true;
00421     } else {
00422         return false;
00423     }
00424 }
00425
00432 bool asm_isdigit(char c)
00433 {
00434     if (c >= '0' && c <= '9') {
00435         return true;
00436     } else {
00437         return false;
00438     }
00439 }
00440
00447 bool asm_isgraph(char c)
00448 {
00449     if (c >= 33 && c <= 126) {
00450         return true;
00451     } else {
00452         return false;
00453     }
00454 }
00455
00462 bool asm_islower(char c)
00463 {
00464     if (c >= 'a' && c <= 'z') {
00465         return true;
00466     } else {
00467         return false;
00468     }
00469 }
00470
00478 bool asm_isprint(char c)
00479 {
00480     return asm_isgraph(c) || c == ' ';
00481 }
00482
00490 bool asm ispunct(char c)
00491 {
00492     if ((c >= 33 && c <= 47) || (c >= 58 && c <= 64) || (c >= 91 && c <= 96) || (c >= 123 && c <=
        126)) {
00493         return true;
00494     } else {
00495         return false;
00496     }
00497 }
00498
00506 bool asm_isspace(char c)
00507 {
00508     if (c == ' ' || c == '\n' || c == '\t' ||
00509         c == '\v' || c == '\f' || c == '\r') {

```

```

00510         return true;
00511     } else {
00512         return false;
00513     }
00514 }
00515
00522 bool asm_isupper(char c)
00523 {
00524     if (c >= 'A' && c <= 'Z') {
00525         return true;
00526     } else {
00527         return false;
00528     }
00529 }
00530
00537 bool asm_isxdigit(char c)
00538 {
00539     if ((c >= 'a' && c <= 'f') || asm_isdigit(c)) {
00540         return true;
00541     } else {
00542         return false;
00543     }
00544 }
00545
00552 bool asm_isXdigit(char c)
00553 {
00554     if ((c >= 'A' && c <= 'F') || asm_isdigit(c)) {
00555         return true;
00556     } else {
00557         return false;
00558     }
00559 }
00560
00575 void asm_left_pad(char * const s, const size_t padding, const char pad)
00576 {
00577     int len = (int)asm_length(s);
00578     for (int i = len; i >= 0; i--) {
00579         s[i+(int)padding] = s[i];
00580     }
00581     for (int i = 0; i < (int)padding; i++) {
00582         s[i] = pad;
00583     }
00584 }
00585
00599 void asm_left_shift(char * const s, const size_t shift)
00600 {
00601     size_t len = asm_length(s);
00602
00603     if (shift == 0) return;
00604     if (len <= shift) {
00605         s[0] = '\0';
00606         return;
00607     }
00608
00609     size_t i;
00610     for (i = shift; i < len; i++) {
00611         s[i-shift] = s[i];
00612     }
00613     s[i-shift] = '\0';
00614 }
00615
00626 size_t asm_length(const char * const str)
00627 {
00628     char c;
00629     size_t i = 0;
00630
00631     while ((c = str[i++]) != '\0') {
00632         if (i > ASM_MAX_LEN) {
00633             #ifndef NO_ERRORS
00634             fprintf(stderr, "%s:%d:\n%s:\n[Error] index exceeds ASM_MAX_LEN_LINE. Probably no NULL
termination.\n\n", __FILE__, __LINE__, __func__);
00635             #endif
00636             return __SIZE_MAX__;
00637         }
00638     }
00639     return --i;
00640 }
00641
00661 void * asm_memset(void * const des, const unsigned char value, const size_t n)
00662 {
00663     unsigned char *ptr = (unsigned char *)des;
00664     for (size_t i = n; i-- > 0;) {
00665         *ptr++ = value;
00666     }
00667     return des;
00668 }
00669

```

```

00676 void asm_print_many_times(const char * const str, const size_t n)
00677 {
00678     for (size_t i = 0; i < n; i++) {
00679         printf("%s", str);
00680     }
00681     printf("\n");
00682 }
00683
00696 void asm_remove_char_form_string(char * const s, const size_t index)
00697 {
00698     size_t len = asm_length(s);
00699     if (len == 0) return;
00700     if (index >= len) {
00701         #ifndef NO_ERRORS
00702         fprintf(stderr, "%s:%d:\n%s:\n[Error] index exceeds array length.\n\n", __FILE__, __LINE__,
__func__);
00703         #endif
00704         return;
00705     }
00706     for (size_t i = index; i < len; i++) {
00707         s[i] = s[i+1];
00708     }
00709 }
00710 }
00711
00726 int asm_str_in_str(const char * const src, const char * const word_to_search)
00727 {
00728     int i = 0, num_of_accu = 0;
00729     while (src[i] != '\0') {
00730         if (asm_strncmp(src+i, word_to_search, asm_length(word_to_search))) {
00731             num_of_accu++;
00732         }
00733         i++;
00734     }
00735     return num_of_accu;
00736 }
00737
00756 double asm_str2double(const char * const s, const char ** const end, const size_t base)
00757 {
00758     if (base < 2 || base > 36) {
00759         #ifndef NO_ERRORS
00760         fprintf(stderr, "%s:%d:\n%s:\n[Error] Supported bases are [2...36]. Input: %zu\n\n", __FILE__,
__LINE__, __func__, base);
00761         #endif
00762         if (end) *end = s;
00763         return 0.0f;
00764     }
00765
00766     int num_of_whitespace = 0;
00767     while (asm_isspace(s[num_of_whitespace])) {
00768         num_of_whitespace++;
00769     }
00770
00771     int i = 0;
00772     if (s[0+num_of_whitespace] == '-' || s[0+num_of_whitespace] == '+') {
00773         i++;
00774     }
00775     int sign = s[0+num_of_whitespace] == '-' ? -1 : 1;
00776
00777     size_t left = 0;
00778     for (; asm_check_char_belong_to_base(s[i+num_of_whitespace], base); i++) {
00779         left = base * left + asm_get_char_value_in_base(s[i+num_of_whitespace], base);
00780     }
00781     if (s[i+num_of_whitespace] != '.') {
00782         if (end) *end = s + i + num_of_whitespace;
00783         return (left * sign);
00784     }
00785
00786     i++; /* skip the point */
00787
00788     double right = 0;
00789     size_t divider = base;
00790     for (; asm_check_char_belong_to_base(s[i+num_of_whitespace], base); i++) {
00791         right = right + asm_get_char_value_in_base(s[i+num_of_whitespace], base) / (double)divider;
00792         divider *= base;
00793     }
00794
00795     if (end) *end = s + i + num_of_whitespace;
00796
00797     return sign * (left + right);
00798 }
00799
00818 float asm_str2float(const char * const s, const char ** const end, const size_t base)
00819 {
00820     if (base < 2 || base > 36) {
00821         #ifndef NO_ERRORS
00822         fprintf(stderr, "%s:%d:\n%s:\n[Error] Supported bases are [2...36]. Input: %zu\n\n", __FILE__,

```

```

__LINE__, __func__, base);
00823     #endif
00824     if (end) *end = s;
00825     return 0.0f;
00826 }
00827 int num_of_whitespace = 0;
00828 while (asm_isspace(s[num_of_whitespace])) {
00829     num_of_whitespace++;
00830 }
00831
00832 int i = 0;
00833 if (s[0+num_of_whitespace] == '-' || s[0+num_of_whitespace] == '+') {
00834     i++;
00835 }
00836 int sign = s[0+num_of_whitespace] == '-' ? -1 : 1;
00837
00838 int left = 0;
00839 for (; asm_check_char_belong_to_base(s[i+num_of_whitespace], base); i++) {
00840     left = base * left + asm_get_char_value_in_base(s[i+num_of_whitespace], base);
00841 }
00842 if (s[i+num_of_whitespace] != '.') {
00843     if (end) *end = s + i + num_of_whitespace;
00844     return left * sign;
00845 }
00846
00847 i++; /* skip the point */
00848
00849 float right = 0;
00850 size_t divider = base;
00851 for (; asm_check_char_belong_to_base(s[i+num_of_whitespace], base); i++) {
00852     right = right + asm_get_char_value_in_base(s[i+num_of_whitespace], base) / (float)divider;
00853     divider *= base;
00854 }
00855
00856 if (end) *end = s + i + num_of_whitespace;
00857
00858 return sign * (left + right);
00859 }
00860
00877 int asm_str2int(const char * const s, const char ** const end, const size_t base)
00878 {
00879     if (base < 2 || base > 36) {
00880         #ifndef NO_ERRORS
00881             fprintf(stderr, "%s:%d:%n%s:\n[Error] Supported bases are [2...36]. Input: %zu\n\n", __FILE__,
__LINE__, __func__, base);
00882         #endif
00883         if (end) *end = s;
00884         return 0;
00885     }
00886     int num_of_whitespace = 0;
00887     while (asm_isspace(s[num_of_whitespace])) {
00888         num_of_whitespace++;
00889     }
00890
00891     int n = 0, i = 0;
00892     if (s[0+num_of_whitespace] == '-' || s[0+num_of_whitespace] == '+') {
00893         i++;
00894     }
00895     int sign = s[0+num_of_whitespace] == '-' ? -1 : 1;
00896
00897     for (; asm_check_char_belong_to_base(s[i+num_of_whitespace], base); i++) {
00898         n = base * n + asm_get_char_value_in_base(s[i+num_of_whitespace], base);
00899     }
00900
00901     if (end) *end = s + i+num_of_whitespace;
00902
00903     return n * sign;
00904 }
00905
00922 size_t asm_str2size_t(const char * const s, const char ** const end, const size_t base)
00923 {
00924     if (end) *end = s;
00925
00926     int num_of_whitespace = 0;
00927     while (asm_isspace(s[num_of_whitespace])) {
00928         num_of_whitespace++;
00929     }
00930
00931     if (s[0+num_of_whitespace] == '-') {
00932         #ifndef NO_ERRORS
00933             fprintf(stderr, "%s:%d:%n%s:\n[Error] Unable to convert a negative number to size_t.\n\n",
__FILE__, __LINE__, __func__);
00934         #endif
00935         return 0;
00936     }
00937
00938     if (base < 2 || base > 36) {

```



```

00939     #ifndef NO_ERRORS
00940     fprintf(stderr, "%s:%d:\n%s:\n[Error] Supported bases are [2...36]. Input: %zu\n\n", __FILE__,
__LINE__, __func__, base);
00941     #endif
00942     if (end) *end = s+num_of_whitespace;
00943     return 0;
00944 }
00945
00946     size_t n = 0, i = 0;
00947     if (s[0+num_of_whitespace] == '+') {
00948         i++;
00949     }
00950
00951     for (; asm_check_char_belong_to_base(s[i+num_of_whitespace], base); i++) {
00952         n = base * n + asm_get_char_value_in_base(s[i+num_of_whitespace], base);
00953     }
00954
00955     if (end) *end = s + i+num_of_whitespace;
00956
00957     return n;
00958 }
00959
00969 void asm_strip_whitespace(char * const s)
00970 {
00971     size_t len = asm_length(s);
00972     size_t i;
00973     for (i = 0; i < len; i++) {
00974         if (asm_isspace(s[i])) {
00975             asm_remove_char_from_string(s, i);
00976             len--;
00977             i--;
00978         }
00979     }
00980     s[i] = '\0';
00981 }
00982
00990 bool asm_str_is_whitespace(const char * const s)
00991 {
00992     for (size_t i = 0; i < asm_length(s); i++) {
00993         if (!asm_isspace(s[i])) {
00994             return false;
00995         }
00996     }
00997
00998     return true;
00999 }
01000
01021 int asm_strncat(char * const s1, const char * const s2, const int N)
01022 {
01023     size_t len_s1 = asm_length(s1);
01024
01025     int limit = N;
01026     if (limit == 0) {
01027         limit = ASM_MAX_LEN;
01028     }
01029
01030     int i = 0;
01031     while (i < limit && s2[i] != '\0') {
01032         if (len_s1 + (size_t)i >= ASM_MAX_LEN) {
01033             #ifndef NO_ERRORS
01034             fprintf(stderr, "%s:%d:\n%s:\n[Error] s2 or the first N=%d digit of s2 does not fit into
s1.\n\n", __FILE__, __LINE__, __func__, N);
01035             #endif
01036             return i;
01037         }
01038
01039         s1[len_s1+i] = s2[i];
01040         i++;
01041     }
01042     return i;
01043 }
01044
01061 int asm_strncmp(const char *s1, const char *s2, const int N)
01062 {
01063     int i = 0;
01064     while (i < N) {
01065         if (s1[i] == '\0' && s2[i] == '\0') {
01066             break;
01067         }
01068         if (s1[i] != s2[i] || (s1[i] == '\0') || (s2[i] == '\0')) {
01069             return 0;
01070         }
01071         i++;
01072     }
01073     return 1;
01074 }
01075

```

```

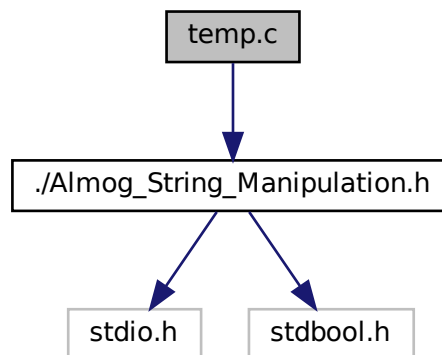
01081 void asm_tolower(char * const s)
01082 {
01083     size_t len = asm_length(s);
01084     for (size_t i = 0; i < len; i++) {
01085         if (asm_isupper(s[i])) {
01086             s[i] += 'a' - 'A';
01087         }
01088     }
01089 }
01090
01096 void asm_toupper(char * const s)
01097 {
01098     size_t len = asm_length(s);
01099     for (size_t i = 0; i < len; i++) {
01100         if (asm_islower(s[i])) {
01101             s[i] += 'A' - 'a';
01102         }
01103     }
01104 }
01105
01106 #ifdef NO_ERRORS
01107 #undef NO_ERRORS
01108 #endif
01109
01110 #endif /*ALMOG_STRING_MANIPULATION_IMPLEMENTATION*/
01111

```

2.3 temp.c File Reference

#include "Almog_String_Manipulation.h"

Include dependency graph for temp.c:



Macros

- #define `ALMOG_STRING_MANIPULATION_IMPLEMENTATION`

Functions

- int `main` (void)

2.3.1 Macro Definition Documentation

2.3.1.1 ALMOG_STRING_MANIPULATION_IMPLEMENTATION

```
#define ALMOG_STRING_MANIPULATION_IMPLEMENTATION
```

Definition at line 1 of file [temp.c](#).

2.3.2 Function Documentation

2.3.2.1 main()

```
int main (  
    void )
```

Definition at line 4 of file [temp.c](#).

References [asm_copy_array_by_indexes\(\)](#), [asm_dprintSTRING](#), [asm_get_token_and_cut\(\)](#), [asm_left_pad\(\)](#), and [asm_length\(\)](#).

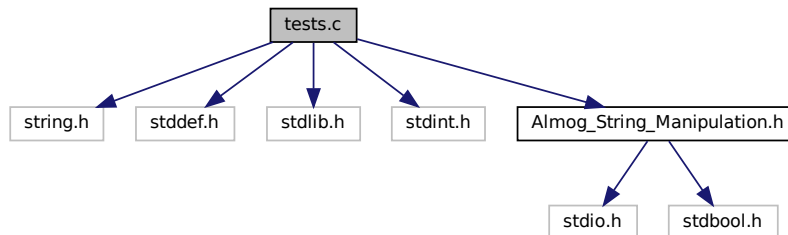
2.4 temp.c

```
00001 #define ALMOG_STRING_MANIPULATION_IMPLEMENTATION  
00002 #include "Almog_String_Manipulation.h"  
00003  
00004 int main(void)  
00005 {  
00006     char str[] = "1012";  
00007     char str1[5] = {'2'};  
00008  
00009     asm_copy_array_by_indexes(str1, 0, asm_length(str), str);  
00010  
00011     asm_dprintSTRING(str);  
00012     asm_dprintSTRING(str1);  
00013  
00014     asm_get_token_and_cut(str1, str, '0', false);  
00015  
00016     asm_dprintSTRING(str);  
00017     asm_dprintSTRING(str1);  
00018  
00019     asm_left_pad(str1, 2, '*');  
00020     asm_dprintSTRING(str1);  
00021  
00022  
00023     return 0;  
00024 }
```

2.5 tests.c File Reference

```
#include <string.h>
#include <stddef.h>
#include <stdlib.h>
#include <stdint.h>
#include "Almog_String_Manipulation.h"
```

Include dependency graph for tests.c:



Macros

- `#define` [ALMOG_STRING_MANIPULATION_IMPLEMENTATION](#)
- `#define` [NO_ERRORS](#)
- `#define` [TEST_CASE](#)(expr)
- `#define` [TEST_WARN](#)(expr, msg)
- `#define` [TEST_EQ_INT](#)(a, b) [TEST_CASE](#)((a) == (b))
- `#define` [TEST_EQ_SIZE](#)(a, b) [TEST_CASE](#)((a) == (b))
- `#define` [TEST_EQ_STR](#)(a, b) [TEST_CASE](#)(strcmp((a), (b)) == 0)
- `#define` [TEST_NE_STR](#)(a, b) [TEST_CASE](#)(strcmp((a), (b)) != 0)

Functions

- static void [fill_sentinel](#) (unsigned char *buf, size_t n, unsigned char v)
- static bool [is_nul_terminated_within](#) (const char *s, size_t cap)
- static uint32_t [xorshift32](#) (void)
- static char [rand_ascii_printable](#) (void)
- static void [test_ascii_classification_exhaustive_ranges](#) (void)
- static void [test_ascii_classification_full_scan_0_127](#) (void)
- static void [test_case_conversion_roundtrip](#) (void)
- static void [test_length_matches_strlen_small](#) (void)
- static void [test_memset_basic_and_edges](#) (void)
- static void [test_copy_array_by_indexes_behavior_and_bounds](#) (void)
- static void [test_left_shift_edges](#) (void)
- static void [test_left_pad_edges_and_sentinel](#) (void)
- static void [test_remove_char_form_string_edges](#) (void)
- static void [test_strip_whitespace_properties](#) (void)
- static void [test_str_is_whitespace_edges](#) (void)
- static void [test_strncmp_boolean_edges](#) (void)
- static void [test_str_in_str_overlap_and_edges](#) (void)

- static void [test_base_digit_helpers](#) (void)
- static void [test_str2int](#) (void)
- static void [test_str2size_t](#) (void)
- static void [test_str2float_double](#) (void)
- static void [test_get_next_word_from_line_current_behavior](#) (void)
- static void [test_get_word_and_cut_edges](#) (void)
- static void [test_get_line_tmpfile](#) (void)
- static void [test_get_line_too_long](#) (void)
- static void [test_strncat_current_behavior_and_sentinel](#) (void)
- int [main](#) (void)

Variables

- static int [g_tests_run](#) = 0
- static int [g_tests_failed](#) = 0
- static int [g_tests_warned](#) = 0
- static uint32_t [rng_state](#) = 0xC0FFEE01u

2.5.1 Macro Definition Documentation

2.5.1.1 ALMOG_STRING_MANIPULATION_IMPLEMENTATION

```
#define ALMOG_STRING_MANIPULATION_IMPLEMENTATION
```

Definition at line 9 of file [tests.c](#).

2.5.1.2 NO_ERRORS

```
#define NO_ERRORS
```

Definition at line 10 of file [tests.c](#).

2.5.1.3 TEST_CASE

```
#define TEST_CASE(  
    expr )
```

Value:

```
do {  
    g_tests_run++;  
    if (!(expr)) {  
        g_tests_failed++;  
        fprintf(stderr, "[FAIL] %s:%d: %s\n", __FILE__, __LINE__, #expr);  
    }  
} while (0)
```

Definition at line 19 of file [tests.c](#).

2.5.1.4 TEST_EQ_INT

```
#define TEST_EQ_INT(  
    a,  
    b ) TEST_CASE((a) == (b))
```

Definition at line 38 of file [tests.c](#).

2.5.1.5 TEST_EQ_SIZE

```
#define TEST_EQ_SIZE(  
    a,  
    b ) TEST_CASE((a) == (b))
```

Definition at line 39 of file [tests.c](#).

2.5.1.6 TEST_EQ_STR

```
#define TEST_EQ_STR(  
    a,  
    b ) TEST_CASE(strcmp((a), (b)) == 0)
```

Definition at line 40 of file [tests.c](#).

2.5.1.7 TEST_NE_STR

```
#define TEST_NE_STR(  
    a,  
    b ) TEST_CASE(strcmp((a), (b)) != 0)
```

Definition at line 41 of file [tests.c](#).

2.5.1.8 TEST_WARN

```
#define TEST_WARN(  
    expr,  
    msg )
```

Value:

```
do {  
    g_tests_run++;  
    if (!(expr)) {  
        g_tests_warned++;  
        fprintf(stderr, "[WARN] %s:%d: %s | %s\n", __FILE__, __LINE__,  
            #expr, msg);  
    }  
} while (0)
```

Definition at line 28 of file [tests.c](#).

2.5.2 Function Documentation

2.5.2.1 fill_sentinel()

```
static void fill_sentinel (
    unsigned char * buf,
    size_t n,
    unsigned char v ) [static]
```

Definition at line 43 of file [tests.c](#).

Referenced by [test_copy_array_by_indexes_behavior_and_bounds\(\)](#), [test_get_line_too_long\(\)](#), [test_left_pad_edges_and_sentinel\(\)](#), [test_memset_basic_and_edges\(\)](#), and [test_strncat_current_behavior_and_sentinel\(\)](#).

2.5.2.2 is_nul_terminated_within()

```
static bool is_nul_terminated_within (
    const char * s,
    size_t cap ) [static]
```

Definition at line 48 of file [tests.c](#).

Referenced by [test_case_conversion_roundtrip\(\)](#), [test_get_line_tmpfile\(\)](#), and [test_strncat_current_behavior_and_sentinel\(\)](#).

2.5.2.3 main()

```
int main (
    void )
```

Definition at line 797 of file [tests.c](#).

References [g_tests_failed](#), [g_tests_run](#), [g_tests_warned](#), [test_ascii_classification_exhaustive_ranges\(\)](#), [test_ascii_classification_full_test_base_digit_helpers\(\)](#), [test_case_conversion_roundtrip\(\)](#), [test_copy_array_by_indexes_behavior_and_bounds\(\)](#), [test_get_line_tmpfile\(\)](#), [test_get_line_too_long\(\)](#), [test_get_next_word_from_line_current_behavior\(\)](#), [test_get_word_and_cut_edges\(\)](#), [test_left_pad_edges_and_sentinel\(\)](#), [test_left_shift_edges\(\)](#), [test_length_matches_strlen_small\(\)](#), [test_memset_basic_and_edges\(\)](#), [test_remove_char_from_string_edges\(\)](#), [test_str2float_double\(\)](#), [test_str2int\(\)](#), [test_str2size_t\(\)](#), [test_str_in_str_overlap_and_edges\(\)](#), [test_str_is_whitespace_edges\(\)](#), [test_strip_whitespace_properties\(\)](#), [test_strncat_current_behavior_and_sentinel\(\)](#), and [test_strncmp_boolean_edges\(\)](#).

2.5.2.4 rand_ascii_printable()

```
static char rand_ascii_printable (  
    void ) [static]
```

Definition at line 68 of file [tests.c](#).

References [xorshift32\(\)](#).

Referenced by [test_case_conversion_roundtrip\(\)](#), [test_length_matches_strlen_small\(\)](#), and [test_strip_whitespace_properties\(\)](#).

2.5.2.5 test_ascii_classification_exhaustive_ranges()

```
static void test_ascii_classification_exhaustive_ranges (  
    void ) [static]
```

Definition at line 82 of file [tests.c](#).

References [asm_isalnum\(\)](#), [asm_isalpha\(\)](#), [asm_iscntrl\(\)](#), [asm_isdigit\(\)](#), [asm_isgraph\(\)](#), [asm_islower\(\)](#), [asm_isprint\(\)](#), [asm_ispunct\(\)](#), [asm_isspace\(\)](#), [asm_isupper\(\)](#), [asm_isxdigit\(\)](#), [asm_isXdigit\(\)](#), and [TEST_CASE](#).

Referenced by [main\(\)](#).

2.5.2.6 test_ascii_classification_full_scan_0_127()

```
static void test_ascii_classification_full_scan_0_127 (  
    void ) [static]
```

Definition at line 153 of file [tests.c](#).

References [asm_isalnum\(\)](#), [asm_isalpha\(\)](#), [asm_isdigit\(\)](#), [asm_isgraph\(\)](#), [asm_islower\(\)](#), [asm_isprint\(\)](#), [asm_isupper\(\)](#), and [TEST_CASE](#).

Referenced by [main\(\)](#).

2.5.2.7 test_base_digit_helpers()

```
static void test_base_digit_helpers (  
    void ) [static]
```

Definition at line 460 of file [tests.c](#).

References [asm_check_char_belong_to_base\(\)](#), [asm_get_char_value_in_base\(\)](#), [TEST_CASE](#), and [TEST_EQ_INT](#).

Referenced by [main\(\)](#).

2.5.2.8 test_case_conversion_roundtrip()

```
static void test_case_conversion_roundtrip (  
    void ) [static]
```

Definition at line 181 of file [tests.c](#).

References [asm_tolower\(\)](#), [asm_toupper\(\)](#), [is_nul_terminated_within\(\)](#), [rand_ascii_printable\(\)](#), [TEST_CASE](#), [TEST_EQ_STR](#), and [xorshift32\(\)](#).

Referenced by [main\(\)](#).

2.5.2.9 test_copy_array_by_indexes_behavior_and_bounds()

```
static void test_copy_array_by_indexes_behavior_and_bounds (  
    void ) [static]
```

Definition at line 257 of file [tests.c](#).

References [asm_copy_array_by_indexes\(\)](#), [fill_sentinel\(\)](#), [TEST_CASE](#), and [TEST_EQ_STR](#).

Referenced by [main\(\)](#).

2.5.2.10 test_get_line_tmpfile()

```
static void test_get_line_tmpfile (  
    void ) [static]
```

Definition at line 689 of file [tests.c](#).

References [asm_get_line\(\)](#), [ASM_MAX_LEN](#), [g_tests_warned](#), [is_nul_terminated_within\(\)](#), [TEST_CASE](#), [TEST_EQ_INT](#), and [TEST_EQ_STR](#).

Referenced by [main\(\)](#).

2.5.2.11 test_get_line_too_long()

```
static void test_get_line_too_long (  
    void ) [static]
```

Definition at line 733 of file [tests.c](#).

References [asm_get_line\(\)](#), [ASM_MAX_LEN](#), [fill_sentinel\(\)](#), [g_tests_warned](#), and [TEST_EQ_INT](#).

Referenced by [main\(\)](#).

2.5.2.12 test_get_next_word_from_line_current_behavior()

```
static void test_get_next_word_from_line_current_behavior (
    void ) [static]
```

Definition at line 606 of file [tests.c](#).

References [asm_get_next_token_from_str\(\)](#), [TEST_CASE](#), [TEST_EQ_INT](#), and [TEST_EQ_STR](#).

Referenced by [main\(\)](#).

2.5.2.13 test_get_word_and_cut_edges()

```
static void test_get_word_and_cut_edges (
    void ) [static]
```

Definition at line 651 of file [tests.c](#).

References [asm_get_token_and_cut\(\)](#), [TEST_CASE](#), and [TEST_EQ_STR](#).

Referenced by [main\(\)](#).

2.5.2.14 test_left_pad_edges_and_sentinel()

```
static void test_left_pad_edges_and_sentinel (
    void ) [static]
```

Definition at line 319 of file [tests.c](#).

References [asm_left_pad\(\)](#), [fill_sentinel\(\)](#), [TEST_CASE](#), and [TEST_EQ_STR](#).

Referenced by [main\(\)](#).

2.5.2.15 test_left_shift_edges()

```
static void test_left_shift_edges (
    void ) [static]
```

Definition at line 294 of file [tests.c](#).

References [asm_left_shift\(\)](#), and [TEST_EQ_STR](#).

Referenced by [main\(\)](#).

2.5.2.16 test_length_matches_strlen_small()

```
static void test_length_matches_strlen_small (  
    void ) [static]
```

Definition at line 227 of file [tests.c](#).

References [asm_length\(\)](#), [rand_ascii_printable\(\)](#), [TEST_EQ_SIZE](#), and [xorshift32\(\)](#).

Referenced by [main\(\)](#).

2.5.2.17 test_memset_basic_and_edges()

```
static void test_memset_basic_and_edges (  
    void ) [static]
```

Definition at line 241 of file [tests.c](#).

References [asm_memset\(\)](#), [fill_sentinel\(\)](#), and [TEST_CASE](#).

Referenced by [main\(\)](#).

2.5.2.18 test_remove_char_form_string_edges()

```
static void test_remove_char_form_string_edges (  
    void ) [static]
```

Definition at line 358 of file [tests.c](#).

References [asm_remove_char_form_string\(\)](#), and [TEST_EQ_STR](#).

Referenced by [main\(\)](#).

2.5.2.19 test_str2float_double()

```
static void test_str2float_double (  
    void ) [static]
```

Definition at line 562 of file [tests.c](#).

References [asm_str2double\(\)](#), [asm_str2float\(\)](#), and [TEST_CASE](#).

Referenced by [main\(\)](#).

2.5.2.20 test_str2int()

```
static void test_str2int (
    void ) [static]
```

Definition at line 495 of file [tests.c](#).

References [asm_str2int\(\)](#), and [TEST_CASE](#).

Referenced by [main\(\)](#).

2.5.2.21 test_str2size_t()

```
static void test_str2size_t (
    void ) [static]
```

Definition at line 531 of file [tests.c](#).

References [asm_str2size_t\(\)](#), and [TEST_CASE](#).

Referenced by [main\(\)](#).

2.5.2.22 test_str_in_str_overlap_and_edges()

```
static void test_str_in_str_overlap_and_edges (
    void ) [static]
```

Definition at line 448 of file [tests.c](#).

References [asm_str_in_str\(\)](#), and [TEST_EQ_INT](#).

Referenced by [main\(\)](#).

2.5.2.23 test_str_is_whitespace_edges()

```
static void test_str_is_whitespace_edges (
    void ) [static]
```

Definition at line 423 of file [tests.c](#).

References [asm_str_is_whitespace\(\)](#), and [TEST_CASE](#).

Referenced by [main\(\)](#).

2.5.2.24 test_strip_whitespace_properties()

```
static void test_strip_whitespace_properties (  
    void ) [static]
```

Definition at line 387 of file [tests.c](#).

References [asm_isspace\(\)](#), [asm_strip_whitespace\(\)](#), [rand_ascii_printable\(\)](#), [TEST_CASE](#), [TEST_EQ_STR](#), and [xorshift32\(\)](#).

Referenced by [main\(\)](#).

2.5.2.25 test_strncat_current_behavior_and_sentinel()

```
static void test_strncat_current_behavior_and_sentinel (  
    void ) [static]
```

Definition at line 761 of file [tests.c](#).

References [asm_strncat\(\)](#), [fill_sentinel\(\)](#), [is_nul_terminated_within\(\)](#), [TEST_CASE](#), [TEST_EQ_INT](#), and [TEST_WARN](#).

Referenced by [main\(\)](#).

2.5.2.26 test_strncmp_boolean_edges()

```
static void test_strncmp_boolean_edges (  
    void ) [static]
```

Definition at line 432 of file [tests.c](#).

References [asm_strncmp\(\)](#), and [TEST_CASE](#).

Referenced by [main\(\)](#).

2.5.2.27 xorshift32()

```
static uint32_t xorshift32 (  
    void ) [static]
```

Definition at line 58 of file [tests.c](#).

References [rng_state](#).

Referenced by [rand_ascii_printable\(\)](#), [test_case_conversion_roundtrip\(\)](#), [test_length_matches_strlen_small\(\)](#), and [test_strip_whitespace_properties\(\)](#).

2.5.3 Variable Documentation

2.5.3.1 `g_tests_failed`

```
int g_tests_failed = 0 [static]
```

Definition at line 16 of file [tests.c](#).

Referenced by [main\(\)](#).

2.5.3.2 `g_tests_run`

```
int g_tests_run = 0 [static]
```

Definition at line 15 of file [tests.c](#).

Referenced by [main\(\)](#).

2.5.3.3 `g_tests_warned`

```
int g_tests_warned = 0 [static]
```

Definition at line 17 of file [tests.c](#).

Referenced by [main\(\)](#), [test_get_line_tmpfile\(\)](#), and [test_get_line_too_long\(\)](#).

2.5.3.4 `rng_state`

```
uint32_t rng_state = 0xC0FFEE01u [static]
```

Definition at line 57 of file [tests.c](#).

Referenced by [xorshift32\(\)](#).

2.6 tests.c

```

00001 /* written by AI */
00002 /* test_almog_string_manipulation.c */
00003
00004 #include <string.h>
00005 #include <stddef.h>
00006 #include <stdlib.h>
00007 #include <stdint.h>
00008
00009 #define ALMOG_STRING_MANIPULATION_IMPLEMENTATION
00010 #define NO_ERRORS
00011 #include "Almog_String_Manipulation.h"
00012
00013 /* ----- Test harness ----- */
00014
00015 static int g_tests_run = 0;
00016 static int g_tests_failed = 0;
00017 static int g_tests_warned = 0;
00018
00019 #define TEST_CASE(expr)
00020     do {
00021         g_tests_run++;
00022         if (!(expr)) {
00023             g_tests_failed++;
00024             fprintf(stderr, "[FAIL] %s:%d: %s\n", __FILE__, __LINE__, #expr);
00025         }
00026     } while (0)
00027
00028 #define TEST_WARN(expr, msg)
00029     do {
00030         g_tests_run++;
00031         if (!(expr)) {
00032             g_tests_warned++;
00033             fprintf(stderr, "[WARN] %s:%d: %s | %s\n", __FILE__, __LINE__,
00034                     #expr, msg);
00035         }
00036     } while (0)
00037
00038 #define TEST_EQ_INT(a, b) TEST_CASE((a) == (b))
00039 #define TEST_EQ_SIZE(a, b) TEST_CASE((a) == (b))
00040 #define TEST_EQ_STR(a, b) TEST_CASE(strcmp((a), (b)) == 0)
00041 #define TEST_NE_STR(a, b) TEST_CASE(strcmp((a), (b)) != 0)
00042
00043 static void fill_sentinel(unsigned char *buf, size_t n, unsigned char v)
00044 {
00045     for (size_t i = 0; i < n; i++) buf[i] = v;
00046 }
00047
00048 static bool is_nul_terminated_within(const char *s, size_t cap)
00049 {
00050     for (size_t i = 0; i < cap; i++) {
00051         if (s[i] == '\0') return true;
00052     }
00053     return false;
00054 }
00055
00056 /* Simple deterministic RNG for fuzz-ish tests */
00057 static uint32_t rng_state = 0xC0FFEE01u;
00058 static uint32_t xorshift32(void)
00059 {
00060     uint32_t x = rng_state;
00061     x ^= x << 13;
00062     x ^= x >> 17;
00063     x ^= x << 5;
00064     rng_state = x;
00065     return x;
00066 }
00067
00068 static char rand_ascii_printable(void)
00069 {
00070     /* printable ASCII range 32..126 */
00071     return (char)(32 + (xorshift32() % 95));
00072 }
00073
00074 /* ----- Coverage checks -----
00075 * We can't reliably "assert all symbols exist" at runtime, but we can at least
00076 * ensure we have tests for every IMPLEMENTED function by calling it at least
00077 * once in this file.
00078 */
00079
00080 /* ----- Tests: ASCII classification ----- */
00081
00082 static void test_ascii_classification_exhaustive_ranges(void)
00083 {
00084     /* Check key boundaries and a few midpoints for each function. */
00085     TEST_CASE(asm_isdigit('0'));

```

```

00086     TEST_CASE(asm_isdigit('9'));
00087     TEST_CASE(!asm_isdigit('/'));
00088     TEST_CASE(!asm_isdigit(':'));
00089
00090     TEST_CASE(asm_isupper('A'));
00091     TEST_CASE(asm_isupper('Z'));
00092     TEST_CASE(!asm_isupper('@'));
00093     TEST_CASE(!asm_isupper(' '));
00094
00095     TEST_CASE(asm_islower('a'));
00096     TEST_CASE(asm_islower('z'));
00097     TEST_CASE(!asm_islower(' '));
00098     TEST_CASE(!asm_islower('{'));
00099
00100     TEST_CASE(asm_isalpha('A'));
00101     TEST_CASE(asm_isalpha('z'));
00102     TEST_CASE(!asm_isalpha('0'));
00103
00104     TEST_CASE(asm_isalnum('A'));
00105     TEST_CASE(asm_isalnum('9'));
00106     TEST_CASE(!asm_isalnum('_'));
00107     TEST_CASE(!asm_isalnum(' '));
00108
00109     TEST_CASE(asm_isspace(' '));
00110     TEST_CASE(asm_isspace('\n'));
00111     TEST_CASE(asm_isspace('\t'));
00112     TEST_CASE(asm_isspace('\r'));
00113     TEST_CASE(asm_isspace('\f'));
00114     TEST_CASE(asm_isspace('\x'));
00115     TEST_CASE(!asm_isspace('X'));
00116
00117     TEST_CASE(asm_isgraph('!'));
00118     TEST_CASE(asm_isgraph('~'));
00119     TEST_CASE(!asm_isgraph(' '));
00120
00121     TEST_CASE(asm_isprint(' '));
00122     TEST_CASE(asm_isprint('!'));
00123     TEST_CASE(!asm_isprint('\n'));
00124
00125     TEST_CASE(asm_ispunct('!'));
00126     TEST_CASE(asm_ispunct('/'));
00127     TEST_CASE(asm_ispunct(':'));
00128     TEST_CASE(!asm_ispunct('A'));
00129     TEST_CASE(!asm_ispunct('0'));
00130     TEST_CASE(!asm_ispunct(' '));
00131
00132     TEST_CASE(asm_iscntrl('\0'));
00133     TEST_CASE(asm_iscntrl('\n'));
00134     TEST_CASE(asm_iscntrl(127));
00135     TEST_CASE(!asm_iscntrl('A'));
00136
00137     /* Hex digit helpers (your impl splits by case) */
00138     TEST_CASE(asm_isxdigit('0'));
00139     TEST_CASE(asm_isxdigit('9'));
00140     TEST_CASE(asm_isxdigit('a'));
00141     TEST_CASE(asm_isxdigit('f'));
00142     TEST_CASE(!asm_isxdigit('g'));
00143     TEST_CASE(!asm_isxdigit('A'));
00144
00145     TEST_CASE(asm_isXdigit('0'));
00146     TEST_CASE(asm_isXdigit('9'));
00147     TEST_CASE(asm_isXdigit('A'));
00148     TEST_CASE(asm_isXdigit('F'));
00149     TEST_CASE(!asm_isXdigit('G'));
00150     TEST_CASE(!asm_isXdigit('a'));
00151 }
00152
00153 static void test_ascii_classification_full_scan_0_127(void)
00154 {
00155     /* Property checks over ASCII 0..127. */
00156     for (int c = 0; c <= 127; c++) {
00157         char ch = (char)c;
00158
00159         /* isalnum == isalpha || isdigit */
00160         TEST_CASE(asm_isalnum(ch) == (asm_isalpha(ch) || asm_isdigit(ch)));
00161
00162         /* isprint == isgraph || ' ' */
00163         TEST_CASE(asm_isprint(ch) == (asm_isgraph(ch) || ch == ' '));
00164
00165         /* isalpha implies not digit */
00166         if (asm_isalpha(ch)) {
00167             TEST_CASE(!asm_isdigit(ch));
00168         }
00169
00170         /* upper and lower are disjoint */
00171         if (asm_isupper(ch)) TEST_CASE(!asm_islower(ch));
00172         if (asm_islower(ch)) TEST_CASE(!asm_isupper(ch));

```



```

00173
00174     /* graph implies print */
00175     if (asm_isgraph(ch)) TEST_CASE(asm_isprint(ch));
00176 }
00177 }
00178
00179 /* ----- Tests: case conversion ----- */
00180
00181 static void test_case_conversion_roundtrip(void)
00182 {
00183     for (int i = 0; i < 200; i++) {
00184         char s[128];
00185         char a[128];
00186         char b[128];
00187
00188         /* random printable string length 0..40 */
00189         size_t n = (size_t)(xorshift32() % 41);
00190         for (size_t j = 0; j < n; j++) s[j] = rand_ascii_printable();
00191         s[n] = '\0';
00192
00193         strcpy(a, s);
00194         strcpy(b, s);
00195
00196         asm_tolower(a);
00197         asm_toupper(a);
00198         asm_toupper(b);
00199         asm_tolower(b);
00200
00201         /* Not equal generally, but must still be valid strings and stable */
00202         TEST_CASE(is_nul_terminated_within(a, sizeof(a)));
00203         TEST_CASE(is_nul_terminated_within(b, sizeof(b)));
00204
00205         /* toupper(toupper(x)) == toupper(x) */
00206         char u1[128], u2[128];
00207         strcpy(u1, s);
00208         strcpy(u2, s);
00209         asm_toupper(u1);
00210         asm_toupper(u2);
00211         asm_toupper(u2);
00212         TEST_EQ_STR(u1, u2);
00213
00214         /* tolower(tolower(x)) == tolower(x) */
00215         char l1[128], l2[128];
00216         strcpy(l1, s);
00217         strcpy(l2, s);
00218         asm_tolower(l1);
00219         asm_tolower(l2);
00220         asm_tolower(l2);
00221         TEST_EQ_STR(l1, l2);
00222     }
00223 }
00224
00225 /* ----- Tests: asm_length ----- */
00226
00227 static void test_length_matches_strlen_small(void)
00228 {
00229     for (int i = 0; i < 200; i++) {
00230         char s[256];
00231         size_t n = (size_t)(xorshift32() % 200);
00232         for (size_t j = 0; j < n; j++) s[j] = rand_ascii_printable();
00233         s[n] = '\0';
00234
00235         TEST_EQ_SIZE(asm_length(s), strlen(s));
00236     }
00237 }
00238
00239 /* ----- Tests: asm_memset ----- */
00240
00241 static void test_memset_basic_and_edges(void)
00242 {
00243     unsigned char buf[32];
00244     fill_sentinel(buf, sizeof(buf), 0xCC);
00245
00246     void *ret = asm_memset(buf, 0xAB, sizeof(buf));
00247     TEST_CASE(ret == buf);
00248     for (size_t i = 0; i < sizeof(buf); i++) TEST_CASE(buf[i] == 0xAB);
00249
00250     fill_sentinel(buf, sizeof(buf), 0xCC);
00251     asm_memset(buf, 0xAB, 0);
00252     for (size_t i = 0; i < sizeof(buf); i++) TEST_CASE(buf[i] == 0xCC);
00253 }
00254
00255 /* ----- Tests: asm_copy_array_by_indexes ----- */
00256
00257 static void test_copy_array_by_indexes_behavior_and_bounds(void)
00258 {
00259     const char *src = "abcdef";

```

```

00260     char out[16];
00261
00262     asm_copy_array_by_indexes(out, 1, 3, src); /* inclusive end in impl */
00263     TEST_EQ_STR(out, "bcd");
00264
00265     asm_copy_array_by_indexes(out, 0, 0, src);
00266     TEST_EQ_STR(out, "a");
00267
00268     asm_copy_array_by_indexes(out, 5, 5, src);
00269     TEST_EQ_STR(out, "f");
00270
00271     asm_copy_array_by_indexes(out, 0, 6, src); /* copies '\0' too */
00272     TEST_EQ_STR(out, "abcdef");
00273
00274     /* Sentinel around output buffer to detect overwrite beyond out[16] */
00275     struct {
00276         unsigned char pre[8];
00277         char out2[8];
00278         unsigned char post[8];
00279     } box;
00280
00281     fill_sentinel(box.pre, sizeof(box.pre), 0xA5);
00282     fill_sentinel((unsigned char *)box.out2, sizeof(box.out2), 0xCC);
00283     fill_sentinel(box.post, sizeof(box.post), 0x5A);
00284
00285     /* copy "ab" plus '\0' => should fit exactly */
00286     asm_copy_array_by_indexes(box.out2, 0, 1, "ab");
00287     TEST_EQ_STR(box.out2, "ab");
00288     for (size_t i = 0; i < sizeof(box.pre); i++) TEST_CASE(box.pre[i] == 0xA5);
00289     for (size_t i = 0; i < sizeof(box.post); i++) TEST_CASE(box.post[i] == 0x5A);
00290 }
00291
00292 /* ----- Tests: shifting/padding ----- */
00293
00294 static void test_left_shift_edges(void)
00295 {
00296     char s[64];
00297
00298     strcpy(s, "abcdef");
00299     asm_left_shift(s, 0);
00300     TEST_EQ_STR(s, "abcdef");
00301
00302     strcpy(s, "abcdef");
00303     asm_left_shift(s, 1);
00304     TEST_EQ_STR(s, "bcdef");
00305
00306     strcpy(s, "abcdef");
00307     asm_left_shift(s, 5);
00308     TEST_EQ_STR(s, "f");
00309
00310     strcpy(s, "abcdef");
00311     asm_left_shift(s, 6);
00312     TEST_EQ_STR(s, "");
00313
00314     strcpy(s, "abcdef");
00315     asm_left_shift(s, 1000);
00316     TEST_EQ_STR(s, "");
00317 }
00318
00319 static void test_left_pad_edges_and_sentinel(void)
00320 {
00321     {
00322         char s[64] = "abc";
00323         asm_left_pad(s, 0, ' ');
00324         TEST_EQ_STR(s, "abc");
00325     }
00326     {
00327         char s[64] = "abc";
00328         asm_left_pad(s, 4, ' ');
00329         TEST_EQ_STR(s, "    abc");
00330     }
00331     {
00332         char s[64] = "";
00333         asm_left_pad(s, 3, '_');
00334         TEST_EQ_STR(s, "___");
00335     }
00336
00337     /* Sentinel structure: ensure we don't write before start */
00338     struct {
00339         unsigned char pre[8];
00340         char s[32];
00341         unsigned char post[8];
00342     } box;
00343
00344     fill_sentinel(box.pre, sizeof(box.pre), 0x11);
00345     fill_sentinel((unsigned char *)box.s, sizeof(box.s), 0xCC);
00346     fill_sentinel(box.post, sizeof(box.post), 0x22);

```

```

00347
00348     strcpy(box.s, "x");
00349     asm_left_pad(box.s, 5, '0');
00350     TEST_EQ_STR(box.s, "00000x");
00351
00352     for (size_t i = 0; i < sizeof(box.pre); i++) TEST_CASE(box.pre[i] == 0x11);
00353     for (size_t i = 0; i < sizeof(box.post); i++) TEST_CASE(box.post[i] == 0x22);
00354 }
00355
00356 /* ----- Tests: remove/strip/whitespace ----- */
00357
00358 static void test_remove_char_form_string_edges(void)
00359 {
00360     char s[64];
00361
00362     strcpy(s, "abcd");
00363     asm_remove_char_form_string(s, 1);
00364     TEST_EQ_STR(s, "acd");
00365
00366     strcpy(s, "abcd");
00367     asm_remove_char_form_string(s, 0);
00368     TEST_EQ_STR(s, "bcd");
00369
00370     strcpy(s, "abcd");
00371     asm_remove_char_form_string(s, 3);
00372     TEST_EQ_STR(s, "abc");
00373
00374     strcpy(s, "a");
00375     asm_remove_char_form_string(s, 0);
00376     TEST_EQ_STR(s, "");
00377
00378     strcpy(s, "");
00379     asm_remove_char_form_string(s, 0);
00380     TEST_EQ_STR(s, "");
00381
00382     strcpy(s, "abcd");
00383     asm_remove_char_form_string(s, 999);
00384     TEST_EQ_STR(s, "abcd");
00385 }
00386
00387 static void test_strip_whitespace_properties(void)
00388 {
00389     char s[128];
00390
00391     strcpy(s, " a \t b\nc ");
00392     asm_strip_whitespace(s);
00393     TEST_EQ_STR(s, "abc");
00394
00395     strcpy(s, "no_spaces");
00396     asm_strip_whitespace(s);
00397     TEST_EQ_STR(s, "no_spaces");
00398
00399     strcpy(s, " \t\r\n");
00400     asm_strip_whitespace(s);
00401     TEST_EQ_STR(s, "");
00402
00403     /* Property: result has no whitespace chars */
00404     for (int i = 0; i < 100; i++) {
00405         size_t n = (size_t)(xorshift32() % 60);
00406         for (size_t j = 0; j < n; j++) {
00407             /* mix whitespace and printable */
00408             uint32_t r = xorshift32() % 10;
00409             if (r == 0) s[j] = ' ';
00410             else if (r == 1) s[j] = '\n';
00411             else if (r == 2) s[j] = '\t';
00412             else s[j] = rand_ascii_printable();
00413         }
00414         s[n] = '\0';
00415
00416         asm_strip_whitespace(s);
00417         for (size_t k = 0; s[k] != '\0'; k++) {
00418             TEST_CASE(!asm_isspace(s[k]));
00419         }
00420     }
00421 }
00422
00423 static void test_str_is_whitespace_edges(void)
00424 {
00425     TEST_CASE(asm_str_is_whitespace(" \t\r\n") == true);
00426     TEST_CASE(asm_str_is_whitespace("") == true); /* current behavior */
00427     TEST_CASE(asm_str_is_whitespace(" x ") == false);
00428 }
00429
00430 /* ----- Tests: asm_strncmp (boolean) ----- */
00431
00432 static void test_strncmp_boolean_edges(void)
00433 {

```

```

00434     TEST_CASE(asm_strncmp("abc", "abc", 3) == 1);
00435     TEST_CASE(asm_strncmp("abc", "abd", 3) == 0);
00436     TEST_CASE(asm_strncmp("ab", "abc", 3) == 0);
00437     TEST_CASE(asm_strncmp("abc", "ab", 3) == 0);
00438
00439     TEST_CASE(asm_strncmp("abc", "XYZ", 0) == 1);
00440
00441     TEST_CASE(asm_strncmp("", "", 5) == 1);
00442     TEST_CASE(asm_strncmp("", "a", 1) == 0);
00443     TEST_CASE(asm_strncmp("a", "", 1) == 0);
00444 }
00445
00446 /* ----- Tests: asm_str_in_str ----- */
00447
00448 static void test_str_in_str_overlap_and_edges(void)
00449 {
00450     TEST_EQ_INT(asm_str_in_str("aaaa", "aa"), 3);
00451     TEST_EQ_INT(asm_str_in_str("hello world", "lo"), 1);
00452     TEST_EQ_INT(asm_str_in_str("abc", "abcd"), 0);
00453     TEST_EQ_INT(asm_str_in_str("abababa", "aba"), 3);
00454
00455     /* Do not pass empty needle: undefined-ish for your implementation. */
00456 }
00457
00458 /* ----- Tests: base digit helpers ----- */
00459
00460 static void test_base_digit_helpers(void)
00461 {
00462     TEST_CASE(asm_check_char_belong_to_base('0', 2) == true);
00463     TEST_CASE(asm_check_char_belong_to_base('1', 2) == true);
00464     TEST_CASE(asm_check_char_belong_to_base('2', 2) == false);
00465
00466     TEST_CASE(asm_check_char_belong_to_base('9', 10) == true);
00467     TEST_CASE(asm_check_char_belong_to_base('a', 10) == false);
00468
00469     TEST_CASE(asm_check_char_belong_to_base('a', 16) == true);
00470     TEST_CASE(asm_check_char_belong_to_base('f', 16) == true);
00471     TEST_CASE(asm_check_char_belong_to_base('g', 16) == false);
00472     TEST_CASE(asm_check_char_belong_to_base('A', 16) == true);
00473     TEST_CASE(asm_check_char_belong_to_base('F', 16) == true);
00474     TEST_CASE(asm_check_char_belong_to_base('G', 16) == false);
00475
00476     TEST_CASE(asm_check_char_belong_to_base('z', 36) == true);
00477     TEST_CASE(asm_check_char_belong_to_base('Z', 36) == true);
00478
00479     TEST_EQ_INT(asm_get_char_value_in_base('0', 10), 0);
00480     TEST_EQ_INT(asm_get_char_value_in_base('9', 10), 9);
00481     TEST_EQ_INT(asm_get_char_value_in_base('A', 16), 10);
00482     TEST_EQ_INT(asm_get_char_value_in_base('f', 16), 15);
00483     TEST_EQ_INT(asm_get_char_value_in_base('Z', 36), 35);
00484
00485     TEST_EQ_INT(asm_get_char_value_in_base('g', 16), -1);
00486
00487     /* base validity errors should return false / -1 */
00488     TEST_CASE(asm_check_char_belong_to_base('0', 1) == false);
00489     TEST_CASE(asm_check_char_belong_to_base('0', 37) == false);
00490     TEST_EQ_INT(asm_get_char_value_in_base('0', 1), -1);
00491 }
00492
00493 /* ----- Tests: str2int/size_t/float/double ----- */
00494
00495 static void test_str2int(void)
00496 {
00497     const char *end = NULL;
00498
00499     {
00500         char s[] = " -1011zzz";
00501         int v = asm_str2int(s, &end, 2);
00502         TEST_CASE(v == -11);
00503         TEST_CASE(*end == 'z');
00504     }
00505     {
00506         char s[] = "+7ffff!";
00507         int v = asm_str2int(s, &end, 16);
00508         TEST_CASE(v == 0x7fff);
00509         TEST_CASE(*end == '!');
00510     }
00511     {
00512         char s[] = " +0";
00513         int v = asm_str2int(s, &end, 10);
00514         TEST_CASE(v == 0);
00515         TEST_CASE(*end == '\0');
00516     }
00517     {
00518         char s[] = "xyz";
00519         int v = asm_str2int(s, &end, 10);
00520         TEST_CASE(v == 0);

```

```

00521     TEST_CASE(*end == 'x');
00522 }
00523 {
00524     char s[] = "123";
00525     int v = asm_str2int(s, &end, 1);
00526     TEST_CASE(v == 0);
00527     TEST_CASE(end == s);
00528 }
00529 }
00530
00531 static void test_str2size_t(void)
00532 {
00533     const char *end = NULL;
00534
00535     {
00536         char s[] = " +1f!";
00537         size_t v = asm_str2size_t(s, &end, 16);
00538         TEST_CASE(v == 31u);
00539         TEST_CASE(*end == '!');
00540     }
00541     {
00542         char s[] = " -1";
00543         size_t v = asm_str2size_t(s, &end, 10);
00544         TEST_CASE(v == 0);
00545         TEST_CASE(end == s);
00546     }
00547     {
00548         char s[] = " +0009x";
00549         size_t v = asm_str2size_t(s, &end, 10);
00550         TEST_CASE(v == 9u);
00551         TEST_CASE(*end == 'x');
00552     }
00553     {
00554         char s[] = " 123";
00555         size_t v = asm_str2size_t(s, &end, 37);
00556         TEST_CASE(v == 0);
00557         /* current implementation sets *end = s+num_of_whitespace on invalid base */
00558         TEST_CASE(end == s + 2);
00559     }
00560 }
00561
00562 static void test_str2float_double(void)
00563 {
00564     const char *end = NULL;
00565
00566     {
00567         char s[] = " 10.5x";
00568         float v = asm_str2float(s, &end, 10);
00569         TEST_CASE(v > 10.49f && v < 10.51f);
00570         TEST_CASE(*end == 'x');
00571     }
00572     {
00573         char s[] = "-a.bQ";
00574         double v = asm_str2double(s, &end, 16);
00575         TEST_CASE(v < -10.68 && v > -10.70);
00576         TEST_CASE(*end == 'Q');
00577     }
00578     {
00579         char s[] = " 123.";
00580         double v = asm_str2double(s, &end, 10);
00581         TEST_CASE(v > 122.99 && v < 123.01);
00582         TEST_CASE(*end == '\0');
00583     }
00584     {
00585         char s[] = " .5";
00586         double v = asm_str2double(s, &end, 10);
00587         TEST_CASE(v > 0.49 && v < 0.51);
00588         TEST_CASE(*end == '\0');
00589     }
00590     {
00591         char s[] = " -.";
00592         double v = asm_str2double(s, &end, 10);
00593         TEST_CASE(v == 0.0);
00594         TEST_CASE(*end == '\0');
00595     }
00596     {
00597         char s[] = "12.3";
00598         double v = asm_str2double(s, &end, 37);
00599         TEST_CASE(v == 0.0);
00600         TEST_CASE(end == s);
00601     }
00602 }
00603
00604 /* ----- Tests: tokenization helpers ----- */
00605
00606 static void test_get_next_word_from_line_current_behavior(void)
00607 {

```

```

00608     /* Your implementation:
00609     * - does NOT skip whitespace
00610     * - stops only on delimiter or '\0'
00611     * - returns length (j), not consumed index
00612     */
00613     {
00614         char src[] = "abc,def";
00615         char w[64] = {0};
00616         int r = asm_get_next_token_from_str(w, src, ',');
00617         TEST_EQ_INT(r, 3);
00618         TEST_EQ_STR(w, "abc");
00619     }
00620     {
00621         char src[] = ",def";
00622         char w[64] = {0};
00623         int r = asm_get_next_token_from_str(w, src, ',');
00624         TEST_EQ_INT(r, 0);
00625         TEST_EQ_STR(w, "");
00626     }
00627     {
00628         char src[] = " abc,def";
00629         char w[64] = {0};
00630         int r = asm_get_next_token_from_str(w, src, ',');
00631         TEST_EQ_INT(r, 5);
00632         TEST_EQ_STR(w, " abc");
00633     }
00634     {
00635         char src[] = "abc\ndef";
00636         char w[64] = {0};
00637         int r = asm_get_next_token_from_str(w, src, ',');
00638         TEST_EQ_INT(r, (int)strlen(src));
00639         TEST_EQ_STR(w, "abc\ndef");
00640     }
00641
00642     /* Doc mismatch detection (warn, not fail) */
00643     {
00644         char src[] = " abc,def";
00645         char w[64] = {0};
00646         asm_get_next_token_from_str(w, src, ',');
00647         TEST_CASE(strcmp(w, " abc") == 0);
00648     }
00649 }
00650
00651 static void test_get_word_and_cut_edges(void)
00652 {
00653     {
00654         char src[64] = "abc,def";
00655         char w[64] = {0};
00656         int ok = asm_get_token_and_cut(w, src, ',', true);
00657         TEST_CASE(ok == 1);
00658         TEST_EQ_STR(w, "abc");
00659         TEST_EQ_STR(src, ",def");
00660     }
00661     {
00662         char src[64] = "abc,def";
00663         char w[64] = {0};
00664         int ok = asm_get_token_and_cut(w, src, ',', false);
00665         TEST_CASE(ok == 1);
00666         TEST_EQ_STR(w, "abc");
00667         TEST_EQ_STR(src, "def");
00668     }
00669     {
00670         char src[64] = ",def";
00671         char w[64] = {0};
00672         int ok = asm_get_token_and_cut(w, src, ',', true);
00673         TEST_CASE(ok == 0);
00674         TEST_EQ_STR(w, "");
00675         TEST_EQ_STR(src, ",def");
00676     }
00677     {
00678         char src[64] = "nodelem";
00679         char w[64] = {0};
00680         int ok = asm_get_token_and_cut(w, src, ',', false);
00681         TEST_CASE(ok == 1);
00682         TEST_EQ_STR(w, "nodelem");
00683         TEST_EQ_STR(src, "");
00684     }
00685 }
00686
00687 /* ----- Tests: asm_get_line ----- */
00688
00689 static void test_get_line_tmpfile(void)
00690 {
00691     FILE *fp = tmpfile();
00692     if (!fp) {
00693         fprintf(stderr,
00694             "[WARN] tmpfile() unavailable; skipping asm_get_line tests\n");

```

```

00695         g_tests_warned++;
00696         return;
00697     }
00698
00699     fputs("hello\n", fp);
00700     fputs("\n", fp);
00701     fputs("world", fp);
00702     rewind(fp);
00703
00704     {
00705         char line[ASM_MAX_LEN + 1];
00706         int n = asm_get_line(fp, line);
00707         TEST_EQ_INT(n, 5);
00708         TEST_EQ_STR(line, "hello");
00709         TEST_CASE(is_nul_terminated_within(line, sizeof(line)));
00710     }
00711     {
00712         char line[ASM_MAX_LEN + 1];
00713         int n = asm_get_line(fp, line);
00714         TEST_EQ_INT(n, 0);
00715         TEST_EQ_STR(line, "");
00716     }
00717     {
00718         char line[ASM_MAX_LEN + 1];
00719         int n = asm_get_line(fp, line);
00720         TEST_EQ_INT(n, 5);
00721         TEST_EQ_STR(line, "world");
00722     }
00723     {
00724         char line[ASM_MAX_LEN + 1];
00725         int n = asm_get_line(fp, line);
00726         TEST_EQ_INT(n, -1);
00727     }
00728
00729     fclose(fp);
00730 }
00731
00732 /* Optional: test overflow condition using ASM_MAX_LEN+1 chars before '\n' */
00733 static void test_get_line_too_long(void)
00734 {
00735     FILE *fp = tmpfile();
00736     if (!fp) {
00737         fprintf(stderr,
00738             "[WARN] tmpfile() unavailable; skipping long-line test\n");
00739         g_tests_warned++;
00740         return;
00741     }
00742
00743     for (int i = 0; i < ASM_MAX_LEN + 5; i++) fputc('a', fp);
00744     fputc('\n', fp);
00745     rewind(fp);
00746
00747     char line[ASM_MAX_LEN + 1];
00748     fill_sentinel((unsigned char *)line, sizeof(line), 0xCC);
00749
00750     int n = asm_get_line(fp, line);
00751     TEST_EQ_INT(n, -1);
00752
00753     /* On error, your docs say not guaranteed NUL terminated. We only ensure
00754        we didn't write past buffer size (can't directly prove; but at least
00755        array exists). */
00756     fclose(fp);
00757 }
00758
00759 /* ----- Tests: asm_strncat ----- */
00760
00761 static void test_strncat_current_behavior_and_sentinel(void)
00762 {
00763     /* Current impl does NOT append '\0' (bug-like).
00764        We test both:
00765        - it copies correct bytes
00766        - it does not clobber past allowed region
00767        */
00768     struct {
00769         unsigned char pre[8];
00770         char sl[16];
00771         unsigned char post[8];
00772     } box;
00773
00774     fill_sentinel(box.pre, sizeof(box.pre), 0xAA);
00775     fill_sentinel((unsigned char *)box.sl, sizeof(box.sl), 0xCC);
00776     fill_sentinel(box.post, sizeof(box.post), 0xBB);
00777
00778     strcpy(box.sl, "abc");
00779
00780     int n = asm_strncat(box.sl, "DEF", 3);
00781     TEST_EQ_INT(n, 3);

```

```

00782
00783     TEST_CASE(box.s1[0] == 'a' && box.s1[1] == 'b' && box.s1[2] == 'c');
00784     TEST_CASE(box.s1[3] == 'D' && box.s1[4] == 'E' && box.s1[5] == 'F');
00785
00786     /* warn if it *is* NUL terminated (meaning you fixed it) */
00787     TEST_WARN(!is_nul_terminated_within(box.s1, 7),
00788         "asm_strncat appears to NUL-terminate now; update tests to "
00789         "expect \"abcdef\" as a proper C-string.");
00790
00791     for (size_t i = 0; i < sizeof(box.pre); i++) TEST_CASE(box.pre[i] == 0xAA);
00792     for (size_t i = 0; i < sizeof(box.post); i++) TEST_CASE(box.post[i] == 0xBB);
00793 }
00794
00795 /* ----- Main ----- */
00796
00797 int main(void)
00798 {
00799     test_ascii_classification_exhaustive_ranges();
00800     test_ascii_classification_full_scan_0_127();
00801
00802     test_case_conversion_roundtrip();
00803
00804     test_length_matches_strlen_small();
00805
00806     test_memset_basic_and_edges();
00807
00808     test_copy_array_by_indexes_behavior_and_bounds();
00809
00810     test_left_shift_edges();
00811     test_left_pad_edges_and_sentinel();
00812
00813     test_remove_char_from_string_edges();
00814     test_strip_whitespace_properties();
00815     test_str_is_whitespace_edges();
00816
00817     test_strncmp_boolean_edges();
00818     test_str_in_str_overlap_and_edges();
00819
00820     test_base_digit_helpers();
00821     test_str2int();
00822     test_str2size_t();
00823     test_str2float_double();
00824
00825     test_get_next_word_from_line_current_behavior();
00826     test_get_word_and_cut_edges();
00827
00828     test_get_line_tmpfile();
00829     test_get_line_too_long();
00830
00831     test_strncat_current_behavior_and_sentinel();
00832
00833     if (g_tests_failed == 0) {
00834         if (g_tests_warned == 0) {
00835             printf("[OK] %d tests passed\n", g_tests_run);
00836         } else {
00837             printf("[OK] %d tests passed, %d warnings\n", g_tests_run,
00838                 g_tests_warned);
00839         }
00840         return 0;
00841     }
00842
00843     fprintf(stderr, "[FAIL] %d/%d tests failed (%d warnings)\n", g_tests_failed,
00844         g_tests_run, g_tests_warned);
00845     return 1;
00846 }

```


Index

Almog_String_Manipulation.h, [3](#)
asm_check_char_belong_to_base, [10](#)
asm_copy_array_by_indexes, [10](#)
asm_dprintCHAR, [6](#)
asm_dprintDOUBLE, [7](#)
asm_dprintFLOAT, [7](#)
asm_dprintINT, [7](#)
asm_dprintSIZE_T, [8](#)
asm_dprintSTRING, [8](#)
asm_get_char_value_in_base, [11](#)
asm_get_line, [11](#)
asm_get_next_token_from_str, [12](#)
asm_get_token_and_cut, [13](#)
asm_isalnum, [14](#)
asm_isalpha, [14](#)
asm_iscntrl, [15](#)
asm_isdigit, [15](#)
asm_isgraph, [16](#)
asm_islower, [16](#)
asm_isprint, [16](#)
asm_ispunct, [17](#)
asm_isspace, [17](#)
asm_isupper, [18](#)
asm_isXdigit, [19](#)
asm_isxdigit, [18](#)
asm_left_pad, [19](#)
asm_left_shift, [20](#)
asm_length, [20](#)
asm_max, [8](#)
ASM_MAX_LEN, [9](#)
asm_memset, [21](#)
asm_min, [9](#)
asm_print_many_times, [22](#)
asm_remove_char_form_string, [22](#)
asm_str2double, [22](#)
asm_str2float, [23](#)
asm_str2int, [24](#)
asm_str2size_t, [24](#)
asm_str_in_str, [25](#)
asm_str_is_whitespace, [26](#)
asm_strip_whitespace, [26](#)
asm_strncat, [27](#)
asm_strncmp, [27](#)
asm_tolower, [28](#)
asm_toupper, [28](#)
ALMOG_STRING_MANIPULATION_IMPLEMENTATION
temp.c, [37](#)
tests.c, [39](#)
asm_check_char_belong_to_base
Almog_String_Manipulation.h, [10](#)
asm_copy_array_by_indexes
Almog_String_Manipulation.h, [10](#)
asm_dprintCHAR
Almog_String_Manipulation.h, [6](#)
asm_dprintDOUBLE
Almog_String_Manipulation.h, [7](#)
asm_dprintFLOAT
Almog_String_Manipulation.h, [7](#)
asm_dprintINT
Almog_String_Manipulation.h, [7](#)
asm_dprintSIZE_T
Almog_String_Manipulation.h, [8](#)
asm_dprintSTRING
Almog_String_Manipulation.h, [8](#)
asm_get_char_value_in_base
Almog_String_Manipulation.h, [11](#)
asm_get_line
Almog_String_Manipulation.h, [11](#)
asm_get_next_token_from_str
Almog_String_Manipulation.h, [12](#)
asm_get_token_and_cut
Almog_String_Manipulation.h, [13](#)
asm_isalnum
Almog_String_Manipulation.h, [14](#)
asm_isalpha
Almog_String_Manipulation.h, [14](#)
asm_iscntrl
Almog_String_Manipulation.h, [15](#)
asm_isdigit
Almog_String_Manipulation.h, [15](#)
asm_isgraph
Almog_String_Manipulation.h, [16](#)
asm_islower
Almog_String_Manipulation.h, [16](#)
asm_isprint
Almog_String_Manipulation.h, [16](#)
asm_ispunct
Almog_String_Manipulation.h, [17](#)
asm_isspace
Almog_String_Manipulation.h, [17](#)
asm_isupper
Almog_String_Manipulation.h, [18](#)
asm_isXdigit
Almog_String_Manipulation.h, [19](#)
asm_isxdigit
Almog_String_Manipulation.h, [18](#)
asm_left_pad
Almog_String_Manipulation.h, [19](#)

- asm_left_shift
 - Almog_String_Manipulation.h, [20](#)
- asm_length
 - Almog_String_Manipulation.h, [20](#)
- asm_max
 - Almog_String_Manipulation.h, [8](#)
- ASM_MAX_LEN
 - Almog_String_Manipulation.h, [9](#)
- asm_memset
 - Almog_String_Manipulation.h, [21](#)
- asm_min
 - Almog_String_Manipulation.h, [9](#)
- asm_print_many_times
 - Almog_String_Manipulation.h, [22](#)
- asm_remove_char_form_string
 - Almog_String_Manipulation.h, [22](#)
- asm_str2double
 - Almog_String_Manipulation.h, [22](#)
- asm_str2float
 - Almog_String_Manipulation.h, [23](#)
- asm_str2int
 - Almog_String_Manipulation.h, [24](#)
- asm_str2size_t
 - Almog_String_Manipulation.h, [24](#)
- asm_str_in_str
 - Almog_String_Manipulation.h, [25](#)
- asm_str_is_whitespace
 - Almog_String_Manipulation.h, [26](#)
- asm_strip_whitespace
 - Almog_String_Manipulation.h, [26](#)
- asm_strncat
 - Almog_String_Manipulation.h, [27](#)
- asm_strncmp
 - Almog_String_Manipulation.h, [27](#)
- asm_tolower
 - Almog_String_Manipulation.h, [28](#)
- asm_toupper
 - Almog_String_Manipulation.h, [28](#)
- fill_sentinel
 - tests.c, [41](#)
- g_tests_failed
 - tests.c, [48](#)
- g_tests_run
 - tests.c, [48](#)
- g_tests_warned
 - tests.c, [48](#)
- is_nul_terminated_within
 - tests.c, [41](#)
- main
 - temp.c, [37](#)
 - tests.c, [41](#)
- NO_ERRORS
 - tests.c, [39](#)
- rand_ascii_printable
 - tests.c, [41](#)
- rng_state
 - tests.c, [48](#)
- temp.c, [36](#)
 - ALMOG_STRING_MANIPULATION_IMPLEMENTATION, [37](#)
 - main, [37](#)
- test_ascii_classification_exhaustive_ranges
 - tests.c, [42](#)
- test_ascii_classification_full_scan_0_127
 - tests.c, [42](#)
- test_base_digit_helpers
 - tests.c, [42](#)
- TEST_CASE
 - tests.c, [39](#)
- test_case_conversion_roundtrip
 - tests.c, [42](#)
- test_copy_array_by_indexes_behavior_and_bounds
 - tests.c, [43](#)
- TEST_EQ_INT
 - tests.c, [39](#)
- TEST_EQ_SIZE
 - tests.c, [40](#)
- TEST_EQ_STR
 - tests.c, [40](#)
- test_get_line_tmpfile
 - tests.c, [43](#)
- test_get_line_too_long
 - tests.c, [43](#)
- test_get_next_word_from_line_current_behavior
 - tests.c, [43](#)
- test_get_word_and_cut_edges
 - tests.c, [44](#)
- test_left_pad_edges_and_sentinel
 - tests.c, [44](#)
- test_left_shift_edges
 - tests.c, [44](#)
- test_length_matches_strlen_small
 - tests.c, [44](#)
- test_memset_basic_and_edges
 - tests.c, [45](#)
- TEST_NE_STR
 - tests.c, [40](#)
- test_remove_char_form_string_edges
 - tests.c, [45](#)
- test_str2float_double
 - tests.c, [45](#)
- test_str2int
 - tests.c, [45](#)
- test_str2size_t
 - tests.c, [46](#)
- test_str_in_str_overlap_and_edges
 - tests.c, [46](#)
- test_str_is_whitespace_edges
 - tests.c, [46](#)
- test_strip_whitespace_properties
 - tests.c, [46](#)
- test_strncat_current_behavior_and_sentinel

- tests.c, [47](#)
- test_strncmp_boolean_edges
 - tests.c, [47](#)
- TEST_WARN
 - tests.c, [40](#)
- tests.c, [38](#)
 - ALMOG_STRING_MANIPULATION_IMPLEMENTATION,
[39](#)
 - fill_sentinel, [41](#)
 - g_tests_failed, [48](#)
 - g_tests_run, [48](#)
 - g_tests_warned, [48](#)
 - is_nul_terminated_within, [41](#)
 - main, [41](#)
 - NO_ERRORS, [39](#)
 - rand_ascii_printable, [41](#)
 - rng_state, [48](#)
 - test_ascii_classification_exhaustive_ranges, [42](#)
 - test_ascii_classification_full_scan_0_127, [42](#)
 - test_base_digit_helpers, [42](#)
 - TEST_CASE, [39](#)
 - test_case_conversion_roundtrip, [42](#)
 - test_copy_array_by_indexes_behavior_and_bounds,
[43](#)
 - TEST_EQ_INT, [39](#)
 - TEST_EQ_SIZE, [40](#)
 - TEST_EQ_STR, [40](#)
 - test_get_line_tmpfile, [43](#)
 - test_get_line_too_long, [43](#)
 - test_get_next_word_from_line_current_behavior,
[43](#)
 - test_get_word_and_cut_edges, [44](#)
 - test_left_pad_edges_and_sentinel, [44](#)
 - test_left_shift_edges, [44](#)
 - test_length_matches_strlen_small, [44](#)
 - test_memset_basic_and_edges, [45](#)
 - TEST_NE_STR, [40](#)
 - test_remove_char_form_string_edges, [45](#)
 - test_str2float_double, [45](#)
 - test_str2int, [45](#)
 - test_str2size_t, [46](#)
 - test_str_in_str_overlap_and_edges, [46](#)
 - test_str_is_whitespace_edges, [46](#)
 - test_strip_whitespace_properties, [46](#)
 - test_strncat_current_behavior_and_sentinel, [47](#)
 - test_strncmp_boolean_edges, [47](#)
 - TEST_WARN, [40](#)
 - xorshift32, [47](#)
- xorshift32
 - tests.c, [47](#)