Almog String Manipulation

Generated by Doxygen 1.9.1

Chapter 1

File Index

1.1 File List

Here is a list of all files with brief descriptions:

Chapter 2

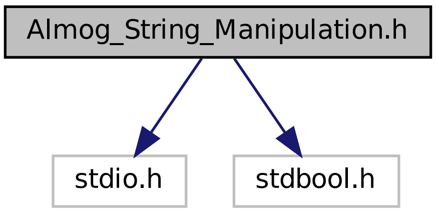
File Documentation

2.1 Almog_String_Manipulation.h File Reference

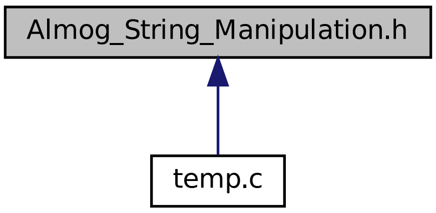
Lightweight string and line manipulation helpers.

```
#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 (char c, size_t base)

  Check if a character is a valid digit in a given base.
- void asm_copy_array_by_indexes (char ∗target, int start, int end, char ∗src)

  Copy a substring [start, end) from src into target and null-terminate.
- size_t asm_get_char_value_in_base (char c)

  Convert a digit character to its numeric value in base-N.
- int asm_get_line (FILE ∗fp, char ∗dst)

  Read a single line from a stream into a buffer.
- int asm_get_next_word_from_line (char ∗dst, char ∗src, char delimiter)

  Extract the next word from a line without modifying the source.
- int asm_get_word_and_cut (char ∗dst, char ∗src, char delimiter, 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 ∗s, size_t padding)

  Left-pad a string with spaces in-place.
- size_t asm_length (char ∗str)

  Compute the length of a null-terminated C string.
- void asm_remove_char_form_string (char ∗s, size_t index)

  Remove a single character from a string by index.
- int asm_str_in_str (char ∗src, char ∗word_to_search)

  Count occurrences of a substring within a string.
- double asm_str2double (char ∗s, char ∗∗end, size_t base)

  Convert a string to double in the given base.
- float asm_str2float (char ∗s, char ∗∗end, size_t base)

  Convert a string to float in the given base.
- int asm_str2int (char ∗s, char ∗∗end, size_t base)

  Convert a string to int in the given base.
- size_t asm_str2size_t (char ∗s, char ∗∗end, size_t base)

  Convert a string to size_t in the given base.
- void asm_strip_whitespace (char ∗s)

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

  Convert all ASCII letters in a string to lowercase in-place.
- void asm_toupper (char ∗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 "word" (token) from a line using a delimiter
- Cutting the extracted word 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

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

Definition at line 78 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

expr	An expression that yields a double. The expression is evaluated exactly once.

Definition at line 105 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

expr	An expression that yields a float. The expression is evaluated exactly once.

Definition at line 96 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

expr	An expression that yields an int. The expression is evaluated exactly once.

Definition at line 87 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

expr	An expression that yields a size_t. The expression is evaluated exactly once.

Definition at line 114 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

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

Definition at line 69 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

a	First value.
b	Second value.

Returns

The larger of `a` and `b`.

Note

Each parameter is evaluated exactly once.

Definition at line 138 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 60 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

a	First value.
b	Second value.

Returns

The smaller of `a` and `b`.

Note

Each parameter is evaluated exactly once.

Definition at line 126 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 (
             char c,
             size_t base )
```

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

Parameters

c	Character to test (e.g., '0'–'9', 'a'–'z', 'A'–'Z').
base	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 186 of file Almog_String_Manipulation.h.

References asm_isdigit().

Referenced by asm_str2double(), asm_str2float(), asm_str2int(), and asm_str2size_t().

2.1.3.2 asm_copy_array_by_indexes()

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

Copy a substring [start, end) from src into target and null-terminate.

Copies characters with indices i = start, start + 1, ..., end - 1 from `src` into `target`, then writes a terminating '\0'.

Parameters

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

Warning

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

Note

This routine supports in-place "left-shift" usage where target == src and start > 0 (used by asm_get_word_↩ and_cut).

Definition at line 219 of file Almog_String_Manipulation.h.

Referenced by asm_get_word_and_cut().

2.1.3.3 asm_get_char_value_in_base()

```
size_t asm_get_char_value_in_base (
            char c )
```

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

Parameters

c	Digit character ('0'–'9', 'a'–'z', 'A'–'Z').

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 238 of file Almog_String_Manipulation.h.

References asm_isdigit(), and asm_isupper().

Referenced by asm_str2double(), asm_str2float(), asm_str2int(), and asm_str2size_t().

2.1.3.4 asm_get_line()

```
int asm_get_line (
            FILE * fp,
            char * 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

fp	Input stream (must be non-NULL).
dst	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 269 of file Almog_String_Manipulation.h.

References ASM_MAX_LEN.

2.1.3.5 asm_get_next_word_from_line()

```
int asm_get_next_word_from_line (
            char * dst,
            char * src,
            char delimiter )
```

Extract the next word from a line without modifying the source.

Skips leading whitespace in `src` (as determined by asm_isspace), then copies characters into `dst` until one of the following is seen:

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

The copied word in `dst` is null-terminated and is never empty on success.

Special case:

- If the very first non-whitespace character in `src` is the delimiter, '
  ', or '\0', that single character is returned as a one-character "word".

Parameters

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

Returns

The number of characters consumed from `src` (i.e., the index of the first unconsumed character).

Return values

-1	No word was found (e.g., only whitespace before a delimiter or end-of-string).

Note

The source buffer is not altered. To both extract and advance/cut the source, see asm_get_word_and_cut().

Definition at line 318 of file Almog_String_Manipulation.h.

References asm_isspace().

Referenced by asm_get_word_and_cut().

2.1.3.6 asm_get_word_and_cut()

```
int asm_get_word_and_cut (
            char * dst,
            char * src,
            char delimiter,
            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

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

Returns

1 if a word was extracted and `src` adjusted, 0 otherwise.

Definition at line 374 of file Almog_String_Manipulation.h.

References asm_copy_array_by_indexes(), asm_get_next_word_from_line(), and asm_length().

2.1.3.7 asm_isalnum()

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

Test for an alphanumeric character (ASCII).

Parameters

c	Character to test.

Returns

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

Definition at line 399 of file Almog_String_Manipulation.h.

References asm_isalpha(), and asm_isdigit().

2.1.3.8 asm_isalpha()

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

Test for an alphabetic character (ASCII).

Parameters

c	Character to test.

Returns

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

Definition at line 410 of file Almog_String_Manipulation.h.

References asm_islower(), and asm_isupper().

Referenced by asm_isalnum().

2.1.3.9 asm_iscntrl()

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

Test for a control character (ASCII).

Parameters

c	Character to test.

Returns

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

Definition at line 421 of file Almog_String_Manipulation.h.

2.1.3.10 asm_isdigit()

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

Test for a decimal digit (ASCII).

Parameters

c	Character to test.

Returns

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

Definition at line 436 of file Almog_String_Manipulation.h.

Referenced by asm_check_char_belong_to_base(), asm_get_char_value_in_base(), asm_isalnum(), asm_isxdigit(), and asm_isXdigit().

2.1.3.11 asm_isgraph()

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

Test for any printable character except space (ASCII).

Parameters

c	Character to test.

Returns

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

Definition at line 451 of file Almog_String_Manipulation.h.

Referenced by asm_isprint().

2.1.3.12 asm_islower()

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

Test for a lowercase letter (ASCII).

Parameters

c	Character to test.

Returns

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

Definition at line 466 of file Almog_String_Manipulation.h.

Referenced by asm_isalpha(), and asm_toupper().

2.1.3.13 asm_isprint()

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

Test for any printable character including space (ASCII).

Parameters

c	Character to test.

Returns

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

Definition at line 482 of file Almog_String_Manipulation.h.

References asm_isgraph().

2.1.3.14 asm_ispunct()

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

Test for a punctuation character (ASCII).

Parameters

c	Character to test.

Returns

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

Definition at line 494 of file Almog_String_Manipulation.h.

2.1.3.15 asm_isspace()

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

Test for a whitespace character (ASCII).

Parameters

c	Character to test.

Returns

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

Definition at line 510 of file Almog_String_Manipulation.h.

Referenced by asm_get_next_word_from_line(), asm_str2double(), asm_str2float(), asm_str2int(), asm_str2size_t(), and asm_strip_whitespace().

2.1.3.16 asm_isupper()

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

Test for an uppercase letter (ASCII).

Parameters

c	Character to test.

Returns

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

Definition at line 526 of file Almog_String_Manipulation.h.

Referenced by asm_get_char_value_in_base(), asm_isalpha(), and asm_tolower().

2.1.3.17 asm_isxdigit()

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

Test for a hexadecimal digit (lowercase or decimal).

Parameters

c	Character to test.

Returns

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

Definition at line 541 of file Almog_String_Manipulation.h.

References asm_isdigit().

2.1.3.18 asm_isXdigit()

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

Test for a hexadecimal digit (uppercase or decimal).

Parameters

c	Character to test.

Returns

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

Definition at line 556 of file Almog_String_Manipulation.h.

References asm_isdigit().

2.1.3.19 asm_left_pad()

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

Left-pad a string with spaces in-place.

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

Parameters

s	String to pad. Modified in-place.
padding	Number of leading spaces 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 578 of file Almog_String_Manipulation.h.

References asm_length().

2.1.3.20 asm_length()

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

Compute the length of a null-terminated C string.

Parameters

str	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 599 of file Almog_String_Manipulation.h.

References ASM_MAX_LEN.

Referenced by asm_get_word_and_cut(), asm_left_pad(), asm_remove_char_form_string(), asm_str_in_str(), asm_strip_whitespace(), asm_tolower(), and asm_toupper().

2.1.3.21 asm_remove_char_form_string()

```
void asm_remove_char_form_string (
             char * s,
             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

s	String to modify in-place. Must be null-terminated.
index	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 625 of file Almog_String_Manipulation.h.

References asm_length().

Referenced by asm_strip_whitespace().

2.1.3.22 asm_str2double()

```
double asm_str2double (
             char * s,
             char ** end,
             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

s	String to convert. Leading ASCII whitespace is skipped.
end	If non-NULL, ∗end is set to point to the first character not used in the conversion.
base	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 683 of file Almog_String_Manipulation.h.

References asm_check_char_belong_to_base(), asm_get_char_value_in_base(), and asm_isspace().

Referenced by main().

2.1.3.23 asm_str2float()

```
float asm_str2float (
            char * s,
            char ** end,
            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

s	String to convert. Leading ASCII whitespace is skipped.
end	If non-NULL, ∗end is set to point to the first character not used in the conversion.
base	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 741 of file Almog_String_Manipulation.h.

References asm_check_char_belong_to_base(), asm_get_char_value_in_base(), and asm_isspace().

Referenced by main().

2.1.3.24 asm_str2int()

```
int asm_str2int (
            char * s,
            char ** end,
            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

s	String to convert. Leading ASCII whitespace is skipped.
end	If non-NULL, ∗end is set to point to the first character not used in the conversion.
base	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 797 of file Almog_String_Manipulation.h.

References asm_check_char_belong_to_base(), asm_get_char_value_in_base(), and asm_isspace().

Referenced by main().

2.1.3.25 asm_str2size_t()

```
size_t asm_str2size_t (
            char * s,
            char ** end,
            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

s	String to convert. Leading ASCII whitespace is skipped.
end	If non-NULL, ∗end is set to point to the first character not used in the conversion.
base	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 839 of file Almog_String_Manipulation.h.

References asm_check_char_belong_to_base(), asm_get_char_value_in_base(), and asm_isspace().

Referenced by main().

2.1.3.26 asm_str_in_str()

```
int asm_str_in_str (
            char * src,
            char * 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

src	The string to search in (must be null-terminated).
word_to_search	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 653 of file Almog_String_Manipulation.h.

References asm_length(), and asm_strncmp().

2.1.3.27 asm_strip_whitespace()

```
void asm_strip_whitespace (
             char * 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

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

Definition at line 879 of file Almog_String_Manipulation.h.

References asm_isspace(), asm_length(), and asm_remove_char_form_string().

2.1.3.28 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

s1	First string (may be shorter than `N`).
s2	Second string (may be shorter than `N`).
N	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 909 of file Almog_String_Manipulation.h.

Referenced by asm_str_in_str().

2.1.3.29 asm_tolower()

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

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

Parameters

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

Definition at line 929 of file Almog_String_Manipulation.h.

References asm_isupper(), and asm_length().

2.1.3.30 asm_toupper()

```
void asm_toupper (
             char * 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 944 of file Almog_String_Manipulation.h.

References asm_islower(), and asm_length().

2.2 Almog_String_Manipulation.h

```

#ifndef ALMOG_STRING_MANIPULATION_H_
#define ALMOG_STRING_MANIPULATION_H_

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

#define ASM_MAX_LEN (int)1e3

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

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

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

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

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

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

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

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

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

```
void    asm_copy_array_by_indexes(char *target, int start, int end, char *src);
size_t  asm_get_char_value_in_base(char c);
int     asm_get_line(FILE *fp, char *dst);
int     asm_get_next_word_from_line(char *dst, char *src, char delimiter);
int     asm_get_word_and_cut(char *dst, char *src, char delimiter, bool leave_delimiter);
bool    asm_isalnum(char c);
bool    asm_isalpha(char c);
bool    asm_iscntrl(char c);
bool    asm_isdigit(char c);
bool    asm_isgraph(char c);
bool    asm_islower(char c);
bool    asm_isprint(char c);
bool    asm_ispunct(char c);
bool    asm_isspace(char c);
bool    asm_isupper(char c);
bool    asm_isxdigit(char c);
bool    asm_isXdigit(char c);
void    asm_left_pad(char *s, size_t padding);
size_t  asm_length(char *str);
void    asm_remove_char_form_string(char *s, size_t index);
int     asm_str_in_str(char *src, char *word_to_search);
double  asm_str2double(char *s, char **end, size_t base);
float   asm_str2float(char *s, char **end, size_t base);
int     asm_str2int(char *s, char **end, size_t base);
size_t  asm_str2size_t(char *s, char **end, size_t base);
void    asm_strip_whitespace(char *s);
int     asm_strncmp(const char *s1, const char *s2, const int N);
void    asm_tolower(char *s);
void    asm_toupper(char *s);

#endif /*ALMOG_STRING_MANIPULATION_H_*/

#ifdef ALMOG_STRING_MANIPULATION_IMPLEMENTATION
#undef ALMOG_STRING_MANIPULATION_IMPLEMENTATION

bool asm_check_char_belong_to_base(char c, size_t base)
{
    if (base > 36 || base < 2) {
        fprintf(stderr, "%s:%d:\n%s:\n[Error] Supported bases are [2...36]. Inputted: %zu\n\n", __FILE__, __LINE__, __func__, base);
        return false;
    }
    if (base <= 10) {
        return c >= '0' && c <= '9'+(char)base-10;
    }
    if (base > 10) {
        return asm_isdigit(c) || (c >= 'A' && c <= ('A'+(char)base-11)) || (c >= 'a' && c <= ('a'+(char)base-11));
    }

    return false;
}

void asm_copy_array_by_indexes(char *target, int start, int end, char *src)
{
    int j = 0;
    for (int i = start; i < end; i++) {
        target[j] = src[i];
        j++;
    }
    target[j] = '\0';
}

size_t asm_get_char_value_in_base(char c)
{
    if (asm_isdigit(c)) {
        return c - '0';
    } else if (asm_isupper(c)) {
        return c - 'A' + 10;
    } else {
        return c - 'a' + 10;
    }
}

int asm_get_line(FILE *fp, char *dst)
{
    int i = 0;
    int c;

    while ((c = fgetc(fp)) != '\n' && c != EOF) {
        dst[i] = c;
        i++;
        if (i >= ASM_MAX_LEN) {
            fprintf(stderr, "%s:%d:\n%s:\n[Error] index exceeds ASM_MAX_LEN. Line in file is too long.\n\n", __FILE__, __LINE__, __func__);
            return -1;
        }
```

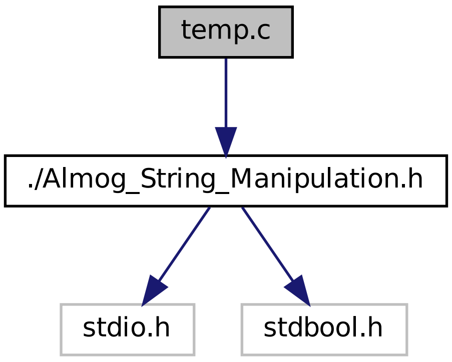
```c
    }
    dst[i] = '\0';
    if (c == EOF && i == 0) {
        return -1;
    }
    return i;
}

int asm_get_next_word_from_line(char *dst, char *src, char delimiter)
{
    int i = 0, j = 0;
    char c;

    while (asm_isspace((c = src[i]))) {
        i++;
    }

    while ((c = src[i]) != delimiter && c != '\n'&& c != '\0') {
        dst[j] = src[i];
        i++;
        j++;
    }

    if ((c == delimiter || c == '\n'|| c == '\0') && i == 0) {
        dst[j++] = c;
        i++;
    }

    dst[j] = '\0';

    if (j == 0) {
        return -1;
    }
    return i;
}

int asm_get_word_and_cut(char *dst, char *src, char delimiter, bool leave_delimiter)
{
    int last_pos;

    if (src[0] == '\0') {
        return 0;
    }
    last_pos = asm_get_next_word_from_line(dst, src, delimiter);
    if (last_pos == -1) {
        return 0;
    }
    if (leave_delimiter) {
        asm_copy_array_by_indexes(src, last_pos, asm_length(src), src);
    } else {
        asm_copy_array_by_indexes(src, last_pos + 1, asm_length(src), src);
    }
    return 1;
}

bool asm_isalnum(char c)
{
    return asm_isalpha(c) || asm_isdigit(c);
}

bool asm_isalpha(char c)
{
    return asm_isupper(c) || asm_islower(c);
}

bool asm_iscntrl(char c)
{
    if ((c >= 0 && c <= 31) || c == 127) {
        return true;
    } else {
        return false;
    }
}

bool asm_isdigit(char c)
{
    if (c >= '0' && c <= '9') {
        return true;
    } else {
        return false;
    }
}

bool asm_isgraph(char c)
{
    if (c >= 33 && c <= 126) {
        return true;
    } else {
        return false;
    }
}

bool asm_islower(char c)
{
    if (c >= 'a' && c <= 'z') {
        return true;
    } else {
        return false;
    }
}

bool asm_isprint(char c)
{
    return asm_isgraph(c) || c == ' ';
}

bool asm_ispunct(char c)
{
    if ((c >= 33 && c <= 47) || (c >= 58 && c <= 64) || (c >= 91 && c <= 96) || (c >= 123 && c <= 126)) {
        return true;
    } else {
        return false;
    }
}

bool asm_isspace(char c)
{
    if (c == ' ' || c == '\n' || c == '\t' ||
        c == '\v'|| c == '\f' || c == '\r') {
        return true;
    } else {
        return false;
    }
}

bool asm_isupper(char c)
{
    if (c >= 'A' && c <= 'Z') {
        return true;
    } else {
        return false;
    }
}

bool asm_isxdigit(char c)
{
    if ((c >= 'a' && c <= 'f') || asm_isdigit(c)) {
        return true;
    } else {
        return false;
    }
}

bool asm_isXdigit(char c)
{
    if ((c >= 'A' && c <= 'F') || asm_isdigit(c)) {
        return true;
    } else {
        return false;
    }
}

void asm_left_pad(char *s, size_t padding)
{
    int len = (int)asm_length(s);
    for (int i = len+1; i >= 0; i--) {
        s[i+(int)padding] = s[i];
    }
    for (int i = 0; i < (int)padding; i++) {
        s[i] = ' ';
    }
}

size_t asm_length(char *str)
{
    char c;
    size_t i = 0;

    while ((c = str[i++]) != '\0') {
        if (i > ASM_MAX_LEN) {
            fprintf(stderr, "%s:%d:\n%s:\n[Error] index exceeds ASM_MAX_LEN_LINE. Probably no NULL termination.\n\n", __FILE__, __LINE__, __func__);
            return __SIZE_MAX__;
        }
    }
    return --i;
}

void asm_remove_char_form_string(char *s, size_t index)
{
    size_t len = asm_length(s);
    if (len == 0) return;
    if (index >= len) {
        fprintf(stderr, "%s:%d:\n%s:\n[Error] index exceeds array length.\n\n", __FILE__, __LINE__,
 __func__);
        return;
    }

    for (size_t i = index; i < len; i++) {
        s[i] = s[i+1];
    }
}

int asm_str_in_str(char *src, char *word_to_search)
{
    int i = 0, num_of_accur = 0;
    while (src[i] != '\0') {
        if (asm_strncmp(src+i, word_to_search, asm_length(word_to_search))) {
            num_of_accur++;
        }
        i++;
    }
    return num_of_accur;
}

double asm_str2double(char *s, char **end, size_t base)
{
    if (base < 2 || base > 36) {
        fprintf(stderr, "%s:%d:\n%s:\n[Error] Supported bases are [2...36]. Input: %zu\n\n", __FILE__,
 __LINE__, __func__, base);
        if (end) *end = s;
        return 0.0f;
    }
    while (asm_isspace(*s)) {
        s++;
    }

    int i = 0;
    if (s[0] == '-' || s[0] == '+') {
        i++;
    }
    int sign = s[0] == '-' ? -1 : 1;

    size_t left = 0;
    for (; asm_check_char_belong_to_base(s[i], base); i++) {
        left = base * left + asm_get_char_value_in_base(s[i]);
    }
    if (s[i] != '.') {
        if (end) *end = s + i;
        return (left * sign);
    }

    i++; /* skip the point */

    double right = 0;
    size_t divider = base;
    for (; asm_check_char_belong_to_base(s[i], base); i++) {
        right = right + asm_get_char_value_in_base(s[i]) / (double)divider;
        divider *= base;
    }

    if (end) *end = s + i;

    return sign * (left + right);
}

float asm_str2float(char *s, char **end, size_t base)
{
    if (base < 2 || base > 36) {
        fprintf(stderr, "%s:%d:\n%s:\n[Error] Supported bases are [2...36]. Input: %zu\n\n", __FILE__,
 __LINE__, __func__, base);
        if (end) *end = s;
        return 0.0f;
    }
    while (asm_isspace(*s)) {
        s++;
    }

    int i = 0;
    if (s[0] == '-' || s[0] == '+') {
        i++;
    }
    int sign = s[0] == '-' ? -1 : 1;

    int left = 0;
    for (; asm_check_char_belong_to_base(s[i], base); i++) {
        left = base * left + asm_get_char_value_in_base(s[i]);
    }
    if (s[i] != '.') {
        if (end) *end = s + i;
        return left * sign;
    }

    i++; /* skip the point */

    float right = 0;
    size_t divider = base;
    for (; asm_check_char_belong_to_base(s[i], base); i++) {
        right = right + asm_get_char_value_in_base(s[i]) / (float)divider;
        divider *= base;
    }

    if (end) *end = s + i;

    return sign * (left + right);
}

int asm_str2int(char *s, char **end, size_t base)
{
    if (base < 2 || base > 36) {
        fprintf(stderr, "%s:%d:\n%s:\n[Error] Supported bases are [2...36]. Input: %zu\n\n", __FILE__,
 __LINE__, __func__, base);
        if (end) *end = s;
        return 0;
    }
    while (asm_isspace(*s)) {
        s++;
    }

    int n = 0, i = 0;
    if (s[0] == '-' || s[0] == '+') {
        i++;
    }
    int sign = s[0] == '-' ? -1 : 1;

    for (; asm_check_char_belong_to_base(s[i], base); i++) {
        n = base * n + asm_get_char_value_in_base(s[i]);
    }

    if (end) *end = s + i;

    return n * sign;
}

size_t asm_str2size_t(char *s, char **end, size_t base)
{
    if (base < 2 || base > 36) {
        fprintf(stderr, "%s:%d:\n%s:\n[Error] Supported bases are [2...36]. Input: %zu\n\n", __FILE__,
 __LINE__, __func__, base);
        if (end) *end = s;
        return 0;
    }
    while (asm_isspace(*s)) {
        s++;
    }

    if (s[0] == '-') {
        fprintf(stderr, "%s:%d:\n%s:\n[Error] Unable to convert a negative number to size_t.\n\n",
 __FILE__, __LINE__, __func__);
        if (end) *end = s;
        return 0;
    }

    size_t n = 0, i = 0;
    if (s[0] == '+') {
        i++;
    }

    for (; asm_check_char_belong_to_base(s[i], base); i++) {
        n = base * n + asm_get_char_value_in_base(s[i]);
    }

    if (end) *end = s + i;

    return n;
}
```

```
void asm_strip_whitespace(char *s)
{
    size_t len = asm_length(s);
    size_t i;
    for (i = 0; i < len; i++) {
        if (asm_isspace(s[i])) {
            asm_remove_char_form_string(s, i);
            len--;
            i--;
        }
    }
    s[i] = '\0';
}

int asm_strncmp(const char *s1, const char *s2, const int N)
{
    int i = 0;
    while (i < N) {
        if (s1[i] == '\0' && s2[i] == '\0') {
            break;
        }
        if (s1[i] != s2[i] || (s1[i] == '\0') || (s2[i] == '\0')) {
            return 0;
        }
        i++;
    }
    return 1;
}

void asm_tolower(char *s)
{
    size_t len = asm_length(s);
    for (size_t i = 0; i < len; i++) {
        if (asm_isupper(s[i])) {
            s[i] += 'a' - 'A';
        }
    }
}

void asm_toupper(char *s)
{
    size_t len = asm_length(s);
    for (size_t i = 0; i < len; i++) {
        if (asm_islower(s[i])) {
            s[i] += 'A' - 'a';
        }
    }
}

#endif /*ALMOG_STRING_MANIPULATION_IMPLEMENTATION*/

```

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_dprintDOUBLE, asm_dprintFLOAT, asm_dprintINT, asm_dprintSIZE_T, asm_dprintSTRING, asm_str2double(), asm_str2float(), asm_str2int(), and asm_str2size_t().

2.4 temp.c

```c
#define ALMOG_STRING_MANIPULATION_IMPLEMENTATION
#include "./Almog_String_Manipulation.h"

int main(void)
{
    char str1[] = "10110110110110110110.101";
    char *str = str1;
    char *temp;

    asm_dprintINT(asm_str2int(str, &temp, 10));
    asm_dprintSTRING(temp);
    asm_dprintSIZE_T(asm_str2size_t(str, &temp, 10));
    asm_dprintSTRING(temp);
    asm_dprintFLOAT(asm_str2float(str, &temp, 10));
    asm_dprintSTRING(temp);
    asm_dprintDOUBLE(asm_str2double(str, &temp, 10));
    asm_dprintSTRING(temp);

    return 0;
}
```

Index