

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	5
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	7
2.1.2.6 asm_dprintSTRING	9
2.1.2.7 asm_max	9
2.1.2.8 ASM_MAX_LEN	10
2.1.2.9 asm_min	10
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()	11
2.1.3.3 asm_get_char_value_in_base()	12
2.1.3.4 asm_get_line()	12
2.1.3.5 asm_get_next_word_from_line()	13
2.1.3.6 asm_get_word_and_cut()	14
2.1.3.7 asm_isalnum()	15
2.1.3.8 asm_isalpha()	15
2.1.3.9 asm_iscntrl()	16
2.1.3.10 asm_isdigit()	16
2.1.3.11 asm_isgraph()	16
2.1.3.12 asm_islower()	17
2.1.3.13 asm_isprint()	17
2.1.3.14 asm_ispunct()	18
2.1.3.15 asm_isspace()	18
2.1.3.16 asm_isupper()	19
2.1.3.17 asm_isxdigit()	19
2.1.3.18 asm_isXdigit()	19
2.1.3.19 asm_left_pad()	20
2.1.3.20 asm_length()	20
2.1.3.21 asm_memset()	21
2.1.3.22 asm_remove_char_form_string()	22
2.1.3.23 asm_str2double()	22
2.1.3.24 asm_str2float()	23
2.1.3.25 asm_str2int()	23

2.1.3.26 asm_str2size_t()	24
2.1.3.27 asm_str_in_str()	25
2.1.3.28 asm_strip_whitespace()	25
2.1.3.29 asm_strncat()	26
2.1.3.30 asm_strncmp()	26
2.1.3.31 asm_tolower()	26
2.1.3.32 asm_toupper()	27
2.2 Almog_String_Manipulation.h	27
2.3 temp.c File Reference	33
2.3.1 Macro Definition Documentation	34
2.3.1.1 ALMOG_STRING_MANIPULATION_IMPLEMENTATION	34
2.3.2 Function Documentation	34
2.3.2.1 main()	34
2.4 temp.c	35
Index	37

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	33

Chapter 2

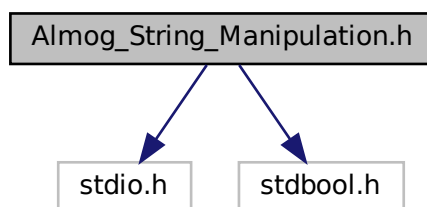
File Documentation

2.1 Almog_String_Manipulation.h File Reference

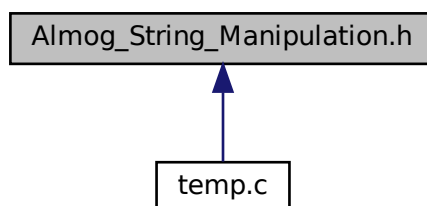
Lightweight string and line manipulation helpers.

```
#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_memset](#) (void *des, unsigned char value, size_t n)
- Set a block of memory to a repeated byte value.*
- void [asm_remove_char_from_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_strncat](#) (char *s1, char *s2, const int N)
- 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 strcmp (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_strcmp` differs from the standard C `strcmp`: 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 80 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 107 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 98 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 89 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 <code>size_t</code> . The expression is evaluated exactly once.
-------------	---

Definition at line 116 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 <code>char</code> (const or non-const). The expression is evaluated exactly once.
-------------	--

Definition at line 71 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 140 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 61 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 128 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

<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 190 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

<i>target</i>	Destination buffer. Must be large enough to hold (end - start) characters plus the null terminator.
<i>start</i>	Inclusive start index within <i>src</i> (0-based).
<i>end</i>	Exclusive end index within <i>src</i> (must satisfy end >= start).
<i>src</i>	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 223 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

<i>c</i>	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 242 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

<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 273 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

<i>dst</i>	Destination buffer for the extracted word. 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 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 322 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

<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 word was extracted and `src` adjusted, 0 otherwise.

Definition at line 378 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

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

Returns

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

Definition at line 403 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

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

Returns

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

Definition at line 414 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

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

Returns

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

Definition at line 425 of file [Almog_String_Manipulation.h](#).

2.1.3.10 asm_isdigit()

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

Test for a decimal digit (ASCII).

Parameters

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

Returns

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

Definition at line 440 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

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

Returns

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

Definition at line 455 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

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

Returns

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

Definition at line 470 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

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

Returns

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

Definition at line 486 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

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

Returns

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

Definition at line 498 of file [Almog_String_Manipulation.h](#).

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', '\n', '\f', or '\r'; false otherwise.

Definition at line 514 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

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

Returns

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

Definition at line 530 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

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

Returns

true if *c* is '0'-'9' or 'a'-'f'; false otherwise.

Definition at line 545 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

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

Returns

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

Definition at line 560 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

<i>s</i>	String to pad. Modified in-place.
<i>padding</i>	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 582 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

<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 603 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_strncat\(\)](#), [asm_tolower\(\)](#), and [asm_toupper\(\)](#).

2.1.3.21 asm_memset()

```
void * asm_memset (
    void * des,
    unsigned char value,
    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 636 of file [Almog_String_Manipulation.h](#).

2.1.3.22 `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

<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 657 of file [Almog_String_Manipulation.h](#).

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

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

2.1.3.23 `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

<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 715 of file [Almog_String_Manipulation.h](#).

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

2.1.3.24 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

<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 773 of file [Almog_String_Manipulation.h](#).

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

2.1.3.25 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

<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 829 of file [Almog_String_Manipulation.h](#).

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

2.1.3.26 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

<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 871 of file [Almog_String_Manipulation.h](#).

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

2.1.3.27 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

<code>src</code>	The string to search in (must be null-terminated).
<code>word_to_search</code>	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 685 of file [Almog_String_Manipulation.h](#).

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

2.1.3.28 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

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

Definition at line 911 of file [Almog_String_Manipulation.h](#).

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

2.1.3.29 `asm_strncat()`

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

Definition at line 925 of file [Almog_String_Manipulation.h](#).

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

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

2.1.3.30 `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

<i>s1</i>	First string (may be shorter than N).
<i>s2</i>	Second string (may be shorter than N).
<i>N</i>	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 963 of file [Almog_String_Manipulation.h](#).

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

2.1.3.31 `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 983 of file [Almog_String_Manipulation.h](#).

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

2.1.3.32 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 998 of file [Almog_String_Manipulation.h](#).

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

2.2 Almog_String_Manipulation.h

```
00001
00039 #ifndef ALMOG_STRING_MANIPULATION_H_
00040 #define ALMOG_STRING_MANIPULATION_H_
00041
00042 #include <stdio.h>
00043 #include <stdbool.h>
00044
00060 #ifndef ASM_MAX_LEN
00061 #define ASM_MAX_LEN (int)1e3
00062 #endif
00063
00071 #define asm_dprintSTRING(expr) printf(#expr " = %s\n", expr)
00072
00080 #define asm_dprintCHAR(expr) printf(#expr " = %c\n", expr)
00081
00089 #define asm_dprintINT(expr) printf(#expr " = %d\n", expr)
00090
00098 #define asm_dprintFLOAT(expr) printf(#expr " = %g\n", expr)
00099
00107 #define asm_dprintDOUBLE(expr) printf(#expr " = %g\n", expr)
00108
00116 #define asm_dprintSIZE_T(expr) printf(#expr " = %zu\n", expr)
00117
00128 #define asm_min(a, b) ((a) < (b) ? (a) : (b))
00129
00140 #define asm_max(a, b) ((a) > (b) ? (a) : (b))
00141
00142 bool    asm_check_char_belong_to_base(char c, size_t base);
00143 void    asm_copy_array_by_indexes(char *target, int start, int end, char *src);
00144 size_t  asm_get_char_value_in_base(char c);
00145 int     asm_get_line(FILE *fp, char *dst);
00146 int     asm_get_next_word_from_line(char *dst, char *src, char delimiter);
00147 int     asm_get_word_and_cut(char *dst, char *src, char delimiter, bool leave_delimiter);
00148 bool    asm_isalnum(char c);
00149 bool    asm_isalpha(char c);
00150 bool    asm_iscntrl(char c);
00151 bool    asm_isdigit(char c);
```

```

00152 bool    asm_isgraph(char c);
00153 bool    asm_islower(char c);
00154 bool    asm_isprint(char c);
00155 bool    asm_isspace(char c);
00156 bool    asm_isupper(char c);
00157 bool    asm_isxdigit(char c);
00158 bool    asm_isXdigit(char c);
00159 void     asm_left_pad(char *s, size_t padding);
00160 size_t   asm_length(char *str);
00161 void *   asm_memset(void *des, unsigned char value, size_t n);
00162 void     asm_remove_char_from_string(char *s, size_t index);
00163 int      asm_str_in_str(char *src, char *word_to_search);
00164 double   asm_str2double(char *s, char **end, size_t base);
00165 float    asm_str2float(char *s, char **end, size_t base);
00166 int      asm_str2int(char *s, char **end, size_t base);
00167 size_t   asm_str2size_t(char *s, char **end, size_t base);
00168 void     asm_strip_whitespace(char *s);
00169 int      asm_strncat(char *s1, char *s2, const int N);
00170 int      asm_strncmp(const char *s1, const char *s2, const int N);
00171 void     asm_tolower(char *s);
00172 void     asm_toupper(char *s);
00173
00174 #endif /*ALMOG_STRING_MANIPULATION_H_*/
00175
00176 #ifdef ALMOG_STRING_MANIPULATION_IMPLEMENTATION
00177 #undef ALMOG_STRING_MANIPULATION_IMPLEMENTATION
00178
00190 bool asm_check_char_belong_to_base(char c, size_t base)
00191 {
00192     if (base > 36 || base < 2) {
00193         fprintf(stderr, "%s:%d:%n%s:\n[Error] Supported bases are [2...36]. Inputted: %zu\n\n",
00194             __FILE__, __LINE__, __func__, base);
00195         return false;
00196     }
00197     if (base <= 10) {
00198         return c >= '0' && c <= '9' + (char)base - 10;
00199     }
00200     if (base > 10) {
00201         return asm_isdigit(c) || (c >= 'A' && c <= ('A' + (char)base - 11)) || (c >= 'a' && c <=
00202             ('a' + (char)base - 11));
00203     }
00204     return false;
00205 }
00223 void asm_copy_array_by_indexes(char *target, int start, int end, char *src)
00224 {
00225     int j = 0;
00226     for (int i = start; i < end; i++) {
00227         target[j] = src[i];
00228         j++;
00229     }
00230     target[j] = '\0';
00231 }
00232
00242 size_t asm_get_char_value_in_base(char c)
00243 {
00244     if (asm_isdigit(c)) {
00245         return c - '0';
00246     } else if (asm_isupper(c)) {
00247         return c - 'A' + 10;
00248     } else {
00249         return c - 'a' + 10;
00250     }
00251 }
00252
00273 int asm_get_line(FILE *fp, char *dst)
00274 {
00275     int i = 0;
00276     int c;
00277
00278     while ((c = fgetc(fp)) != '\n' && c != EOF) {
00279         dst[i] = c;
00280         i++;
00281         if (i >= ASM_MAX_LEN) {
00282             fprintf(stderr, "%s:%d:%n%s:\n[Error] index exceeds ASM_MAX_LEN. Line in file is too
00283             long.\n\n", __FILE__, __LINE__, __func__);
00284             return -1;
00285         }
00286         dst[i] = '\0';
00287         if (c == EOF && i == 0) {
00288             return -1;
00289         }
00290         return i;
00291 }

```

```

00292
00322 int asm_get_next_word_from_line(char *dst, char *src, char delimiter)
00323 {
00324     int i = 0, j = 0;
00325     char c;
00326
00327     while (asm_isspace((c = src[i]))) {
00328         i++;
00329     }
00330
00331     while ((c = src[i]) != delimiter && c != '\n' && c != '\0') {
00332         dst[j] = src[i];
00333         i++;
00334         j++;
00335     }
00336
00337     if ((c == delimiter || c == '\n' || c == '\0') && i == 0) {
00338         dst[j++] = c;
00339         i++;
00340     }
00341
00342     dst[j] = '\0';
00343
00344     if (j == 0) {
00345         return -1;
00346     }
00347     return i;
00348 }
00349
00378 int asm_get_word_and_cut(char *dst, char *src, char delimiter, bool leave_delimiter)
00379 {
00380     int last_pos;
00381
00382     if (src[0] == '\0') {
00383         return 0;
00384     }
00385     last_pos = asm_get_next_word_from_line(dst, src, delimiter);
00386     if (last_pos == -1) {
00387         return 0;
00388     }
00389     if (leave_delimiter) {
00390         asm_copy_array_by_indexes(src, last_pos, asm_length(src), src);
00391     } else {
00392         asm_copy_array_by_indexes(src, last_pos + 1, asm_length(src), src);
00393     }
00394     return 1;
00395 }
00396
00403 bool asm_isalnum(char c)
00404 {
00405     return asm_isalpha(c) || asm_isdigit(c);
00406 }
00407
00414 bool asm_isalpha(char c)
00415 {
00416     return asm_isupper(c) || asm_islower(c);
00417 }
00418
00425 bool asm_iscntrl(char c)
00426 {
00427     if ((c >= 0 && c <= 31) || c == 127) {
00428         return true;
00429     } else {
00430         return false;
00431     }
00432 }
00433
00440 bool asm_isdigit(char c)
00441 {
00442     if (c >= '0' && c <= '9') {
00443         return true;
00444     } else {
00445         return false;
00446     }
00447 }
00448
00455 bool asm_isgraph(char c)
00456 {
00457     if (c >= 33 && c <= 126) {
00458         return true;
00459     } else {
00460         return false;
00461     }
00462 }
00463
00470 bool asm_islower(char c)
00471 {

```

```

00472     if (c >= 'a' && c <= 'z') {
00473         return true;
00474     } else {
00475         return false;
00476     }
00477 }
00478
00486 bool asm_isprint(char c)
00487 {
00488     return asm_isgraph(c) || c == ' ';
00489 }
00490
00498 bool asm_ispunct(char c)
00499 {
00500     if ((c >= 33 && c <= 47) || (c >= 58 && c <= 64) || (c >= 91 && c <= 96) || (c >= 123 && c <=
126)) {
00501         return true;
00502     } else {
00503         return false;
00504     }
00505 }
00506
00514 bool asm_isspace(char c)
00515 {
00516     if (c == ' ' || c == '\n' || c == '\t' ||
00517         c == '\v' || c == '\f' || c == '\r') {
00518         return true;
00519     } else {
00520         return false;
00521     }
00522 }
00523
00530 bool asm_isupper(char c)
00531 {
00532     if (c >= 'A' && c <= 'Z') {
00533         return true;
00534     } else {
00535         return false;
00536     }
00537 }
00538
00545 bool asm_isxdigit(char c)
00546 {
00547     if ((c >= 'a' && c <= 'f') || asm_isdigit(c)) {
00548         return true;
00549     } else {
00550         return false;
00551     }
00552 }
00553
00560 bool asm_isXdigit(char c)
00561 {
00562     if ((c >= 'A' && c <= 'F') || asm_isdigit(c)) {
00563         return true;
00564     } else {
00565         return false;
00566     }
00567 }
00568
00582 void asm_left_pad(char *s, size_t padding)
00583 {
00584     int len = (int)asm_length(s);
00585     for (int i = len+1; i >= 0; i--) {
00586         s[i+(int)padding] = s[i];
00587     }
00588     for (int i = 0; i < (int)padding; i++) {
00589         s[i] = ' ';
00590     }
00591 }
00592
00603 size_t asm_length(char *str)
00604 {
00605     char c;
00606     size_t i = 0;
00607
00608     while ((c = str[i++]) != '\0') {
00609         if (i > ASM_MAX_LEN) {
00610             fprintf(stderr, "%s:%d:\n%s:\n[Error] index exceeds ASM_MAX_LEN_LINE. Probably no NULL
termination.\n\n", __FILE__, __LINE__, __func__);
00611             return __SIZE_MAX__;
00612         }
00613     }
00614     return --i;
00615 }
00616
00636 void * asm_memset(void *des, unsigned char value, size_t n)
00637 {

```

```

00638     unsigned char *ptr = (unsigned char *)des;
00639     while (n-- > 0) {
00640         *ptr++ = value;
00641     }
00642     return des;
00643 }
00644
00657 void asm_remove_char_form_string(char *s, size_t index)
00658 {
00659     size_t len = asm_length(s);
00660     if (len == 0) return;
00661     if (index >= len) {
00662         fprintf(stderr, "%s:%d:\n%s:\n[Error] index exceeds array length.\n\n", __FILE__, __LINE__,
__func__);
00663         return;
00664     }
00665     for (size_t i = index; i < len; i++) {
00666         s[i] = s[i+1];
00667     }
00668 }
00669 }
00670
00685 int asm_str_in_str(char *src, char *word_to_search)
00686 {
00687     int i = 0, num_of_accur = 0;
00688     while (src[i] != '\0') {
00689         if (asm_strncmp(src+i, word_to_search, asm_length(word_to_search))) {
00690             num_of_accur++;
00691         }
00692         i++;
00693     }
00694     return num_of_accur;
00695 }
00696
00715 double asm_str2double(char *s, char **end, size_t base)
00716 {
00717     if (base < 2 || base > 36) {
00718         fprintf(stderr, "%s:%d:\n%s:\n[Error] Supported bases are [2...36]. Input: %zu\n\n", __FILE__,
__LINE__, __func__, base);
00719         if (end) *end = s;
00720         return 0.0f;
00721     }
00722     while (asm_isspace(*s)) {
00723         s++;
00724     }
00725
00726     int i = 0;
00727     if (s[0] == '-' || s[0] == '+') {
00728         i++;
00729     }
00730     int sign = s[0] == '-' ? -1 : 1;
00731
00732     size_t left = 0;
00733     for (; asm_check_char_belong_to_base(s[i], base); i++) {
00734         left = base * left + asm_get_char_value_in_base(s[i]);
00735     }
00736     if (s[i] != '.') {
00737         if (end) *end = s + i;
00738         return (left * sign);
00739     }
00740
00741     i++; /* skip the point */
00742
00743     double right = 0;
00744     size_t divider = base;
00745     for (; asm_check_char_belong_to_base(s[i], base); i++) {
00746         right = right + asm_get_char_value_in_base(s[i]) / (double)divider;
00747         divider *= base;
00748     }
00749
00750     if (end) *end = s + i;
00751
00752     return sign * (left + right);
00753 }
00754
00773 float asm_str2float(char *s, char **end, size_t base)
00774 {
00775     if (base < 2 || base > 36) {
00776         fprintf(stderr, "%s:%d:\n%s:\n[Error] Supported bases are [2...36]. Input: %zu\n\n", __FILE__,
__LINE__, __func__, base);
00777         if (end) *end = s;
00778         return 0.0f;
00779     }
00780     while (asm_isspace(*s)) {
00781         s++;
00782     }
00783

```

```

00784     int i = 0;
00785     if (s[0] == '-' || s[0] == '+') {
00786         i++;
00787     }
00788     int sign = s[0] == '-' ? -1 : 1;
00789
00790     int left = 0;
00791     for (; asm_check_char_belong_to_base(s[i], base); i++) {
00792         left = base * left + asm_get_char_value_in_base(s[i]);
00793     }
00794     if (s[i] != '.') {
00795         if (end) *end = s + i;
00796         return left * sign;
00797     }
00798
00799     i++; /* skip the point */
00800
00801     float right = 0;
00802     size_t divider = base;
00803     for (; asm_check_char_belong_to_base(s[i], base); i++) {
00804         right = right + asm_get_char_value_in_base(s[i]) / (float)divider;
00805         divider *= base;
00806     }
00807
00808     if (end) *end = s + i;
00809
00810     return sign * (left + right);
00811 }
00812
00829 int asm_str2int(char *s, char **end, size_t base)
00830 {
00831     if (base < 2 || base > 36) {
00832         fprintf(stderr, "%s:%d:\n%s:\n[Error] Supported bases are [2...36]. Input: %zu\n", __FILE__,
00833             __LINE__, __func__, base);
00834         if (end) *end = s;
00835         return 0;
00836     }
00837     while (asm_isspace(*s)) {
00838         s++;
00839     }
00840     int n = 0, i = 0;
00841     if (s[0] == '-' || s[0] == '+') {
00842         i++;
00843     }
00844     int sign = s[0] == '-' ? -1 : 1;
00845
00846     for (; asm_check_char_belong_to_base(s[i], base); i++) {
00847         n = base * n + asm_get_char_value_in_base(s[i]);
00848     }
00849
00850     if (end) *end = s + i;
00851
00852     return n * sign;
00853 }
00854
00871 size_t asm_str2size_t(char *s, char **end, size_t base)
00872 {
00873     if (base < 2 || base > 36) {
00874         fprintf(stderr, "%s:%d:\n%s:\n[Error] Supported bases are [2...36]. Input: %zu\n", __FILE__,
00875             __LINE__, __func__, base);
00876         if (end) *end = s;
00877         return 0;
00878     }
00879     while (asm_isspace(*s)) {
00880         s++;
00881     }
00882     if (s[0] == '-') {
00883         fprintf(stderr, "%s:%d:\n%s:\n[Error] Unable to convert a negative number to size_t.\n",
00884             __FILE__, __LINE__, __func__);
00885         if (end) *end = s;
00886         return 0;
00887     }
00888     size_t n = 0, i = 0;
00889     if (s[0] == '+') {
00890         i++;
00891     }
00892
00893     for (; asm_check_char_belong_to_base(s[i], base); i++) {
00894         n = base * n + asm_get_char_value_in_base(s[i]);
00895     }
00896
00897     if (end) *end = s + i;
00898
00899     return n;

```

```

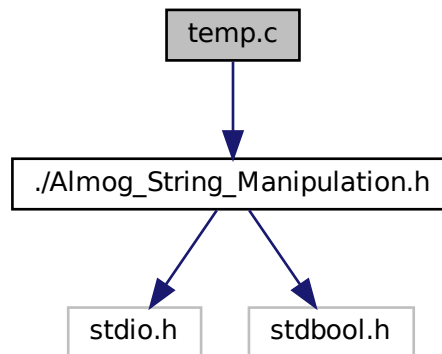
00900 }
00901
00911 void asm_strip_whitespace(char *s)
00912 {
00913     size_t len = asm_length(s);
00914     size_t i;
00915     for (i = 0; i < len; i++) {
00916         if (asm_isspace(s[i])) {
00917             asm_remove_char_from_string(s, i);
00918             len--;
00919             i--;
00920         }
00921     }
00922     s[i] = '\0';
00923 }
00924
00925 int asm_strncat(char *s1, char *s2, const int N)
00926 {
00927     size_t len_s1 = asm_length(s1);
00928
00929     int limit = N;
00930     if (limit == 0) {
00931         limit = ASM_MAX_LEN;
00932     }
00933
00934     int i = 0;
00935     while (i < limit && s2[i] != '\0') {
00936         if (len_s1+i > ASM_MAX_LEN) {
00937             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);
00938             return i;
00939         }
00940
00941         s1[len_s1+i] = s2[i];
00942         i++;
00943     }
00944     return i;
00945 }
00946
00963 int asm_strncmp(const char *s1, const char *s2, const int N)
00964 {
00965     int i = 0;
00966     while (i < N) {
00967         if (s1[i] == '\0' && s2[i] == '\0') {
00968             break;
00969         }
00970         if (s1[i] != s2[i] || (s1[i] == '\0') || (s2[i] == '\0')) {
00971             return 0;
00972         }
00973         i++;
00974     }
00975     return 1;
00976 }
00977
00983 void asm_tolower(char *s)
00984 {
00985     size_t len = asm_length(s);
00986     for (size_t i = 0; i < len; i++) {
00987         if (asm_isupper(s[i])) {
00988             s[i] += 'a' - 'A';
00989         }
00990     }
00991 }
00992
00998 void asm_toupper(char *s)
00999 {
01000     size_t len = asm_length(s);
01001     for (size_t i = 0; i < len; i++) {
01002         if (asm_islower(s[i])) {
01003             s[i] += 'A' - 'a';
01004         }
01005     }
01006 }
01007
01008 #endif /*ALMOG_STRING_MANIPULATION_IMPLEMENTATION*/
01009

```

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_dprintSTRING](#), [ASM_MAX_LEN](#), and [asm_strncat\(\)](#).

2.4 temp.c

```
00001 #define ALMOG_STRING_MANIPULATION_IMPLEMENTATION
00002 #include "./Almog_String_Manipulation.h"
00003
00004 int main(void)
00005 {
00006     char str1[] = "1012";
00007     char str[ASM_MAX_LEN];
00008
00009     asm_dprintSTRING(str);
00010     asm_dprintSTRING(str1);
00011     asm_strncat(str, str1, ASM_MAX_LEN);
00012     asm_dprintSTRING(str);
00013
00014
00015
00016     return 0;
00017 }
```


Index

Almog_String_Manipulation.h, [3](#)
 asm_check_char_belong_to_base, [10](#)
 asm_copy_array_by_indexes, [11](#)
 asm_dprintCHAR, [6](#)
 asm_dprintDOUBLE, [6](#)
 asm_dprintFLOAT, [7](#)
 asm_dprintINT, [7](#)
 asm_dprintSIZE_T, [7](#)
 asm_dprintSTRING, [9](#)
 asm_get_char_value_in_base, [11](#)
 asm_get_line, [12](#)
 asm_get_next_word_from_line, [13](#)
 asm_get_word_and_cut, [14](#)
 asm_isalnum, [15](#)
 asm_isalpha, [15](#)
 asm_iscntrl, [15](#)
 asm_isdigit, [16](#)
 asm_isgraph, [16](#)
 asm_islower, [17](#)
 asm_isprint, [17](#)
 asm_ispunct, [18](#)
 asm_isspace, [18](#)
 asm_isupper, [18](#)
 asm_isXdigit, [19](#)
 asm_isxdigit, [19](#)
 asm_left_pad, [20](#)
 asm_length, [20](#)
 asm_max, [9](#)
 ASM_MAX_LEN, [9](#)
 asm_memset, [21](#)
 asm_min, [10](#)
 asm_remove_char_form_string, [21](#)
 asm_str2double, [22](#)
 asm_str2float, [23](#)
 asm_str2int, [23](#)
 asm_str2size_t, [24](#)
 asm_str_in_str, [24](#)
 asm_strip_whitespace, [25](#)
 asm_strncat, [25](#)
 asm_strncmp, [26](#)
 asm_tolower, [26](#)
 asm_toupper, [27](#)
ALMOG_STRING_MANIPULATION_IMPLEMENTATION
 temp.c, [34](#)
asm_check_char_belong_to_base
 Almog_String_Manipulation.h, [10](#)
asm_copy_array_by_indexes
 Almog_String_Manipulation.h, [11](#)
asm_dprintCHAR
 Almog_String_Manipulation.h, [6](#)
asm_dprintDOUBLE
 Almog_String_Manipulation.h, [6](#)
asm_dprintFLOAT
 Almog_String_Manipulation.h, [7](#)
asm_dprintINT
 Almog_String_Manipulation.h, [7](#)
asm_dprintSIZE_T
 Almog_String_Manipulation.h, [7](#)
asm_dprintSTRING
 Almog_String_Manipulation.h, [9](#)
asm_get_char_value_in_base
 Almog_String_Manipulation.h, [11](#)
asm_get_line
 Almog_String_Manipulation.h, [12](#)
asm_get_next_word_from_line
 Almog_String_Manipulation.h, [13](#)
asm_get_word_and_cut
 Almog_String_Manipulation.h, [14](#)
asm_isalnum
 Almog_String_Manipulation.h, [15](#)
asm_isalpha
 Almog_String_Manipulation.h, [15](#)
asm_iscntrl
 Almog_String_Manipulation.h, [15](#)
asm_isdigit
 Almog_String_Manipulation.h, [16](#)
asm_isgraph
 Almog_String_Manipulation.h, [16](#)
asm_islower
 Almog_String_Manipulation.h, [17](#)
asm_isprint
 Almog_String_Manipulation.h, [17](#)
asm_ispunct
 Almog_String_Manipulation.h, [18](#)
asm_isspace
 Almog_String_Manipulation.h, [18](#)
asm_isupper
 Almog_String_Manipulation.h, [18](#)
asm_isXdigit
 Almog_String_Manipulation.h, [19](#)
asm_isxdigit
 Almog_String_Manipulation.h, [19](#)
asm_left_pad
 Almog_String_Manipulation.h, [20](#)
asm_length
 Almog_String_Manipulation.h, [20](#)
asm_max
 Almog_String_Manipulation.h, [9](#)

ASM_MAX_LEN
 Almog_String_Manipulation.h, [9](#)
asm_memset
 Almog_String_Manipulation.h, [21](#)
asm_min
 Almog_String_Manipulation.h, [10](#)
asm_remove_char_form_string
 Almog_String_Manipulation.h, [21](#)
asm_str2double
 Almog_String_Manipulation.h, [22](#)
asm_str2float
 Almog_String_Manipulation.h, [23](#)
asm_str2int
 Almog_String_Manipulation.h, [23](#)
asm_str2size_t
 Almog_String_Manipulation.h, [24](#)
asm_str_in_str
 Almog_String_Manipulation.h, [24](#)
asm_strip_whitespace
 Almog_String_Manipulation.h, [25](#)
asm_strncat
 Almog_String_Manipulation.h, [25](#)
asm_strncmp
 Almog_String_Manipulation.h, [26](#)
asm_tolower
 Almog_String_Manipulation.h, [26](#)
asm_toupper
 Almog_String_Manipulation.h, [27](#)

main
 temp.c, [34](#)

temp.c, [33](#)
 ALMOG_STRING_MANIPULATION_IMPLEMENTATION,
 [34](#)
 main, [34](#)