

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	6
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	8
2.1.2.7 asm_max	8
2.1.2.8 ASM_MAX_LEN	9
2.1.2.9 asm_min	9
2.1.3 Function Documentation	9
2.1.3.1 asm_check_char_belong_to_base()	9
2.1.3.2 asm_copy_array_by_indexes()	10
2.1.3.3 asm_get_char_value_in_base()	11
2.1.3.4 asm_get_line()	11
2.1.3.5 asm_get_next_word_from_line()	12
2.1.3.6 asm_get_word_and_cut()	13
2.1.3.7 asm_isalnum()	14
2.1.3.8 asm_isalpha()	14
2.1.3.9 asm_iscntrl()	15
2.1.3.10 asm_isdigit()	15
2.1.3.11 asm_isgraph()	15
2.1.3.12 asm_islower()	16
2.1.3.13 asm_isprint()	16
2.1.3.14 asm_ispunct()	17
2.1.3.15 asm_isspace()	17
2.1.3.16 asm_isupper()	18
2.1.3.17 asm_isxdigit()	18
2.1.3.18 asm_isXdigit()	18
2.1.3.19 asm_left_pad()	19
2.1.3.20 asm_length()	19
2.1.3.21 asm_remove_char_form_string()	20
2.1.3.22 asm_str2double()	20
2.1.3.23 asm_str2float()	21
2.1.3.24 asm_str2int()	22
2.1.3.25 asm_str2size_t()	22

2.1.3.26 asm_str_in_str()	23
2.1.3.27 asm_strip_whitespace()	24
2.1.3.28 asm_strncmp()	24
2.1.3.29 asm_tolower()	25
2.1.3.30 asm_toupper()	25
2.2 Almog_String_Manipulation.h	25
2.3 temp.c File Reference	31
2.3.1 Macro Definition Documentation	32
2.3.1.1 ALMOG_STRING_MANIPULATION_IMPLEMENTATION	32
2.3.2 Function Documentation	32
2.3.2.1 main()	32
2.4 temp.c	33
Index	35

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	31

Chapter 2

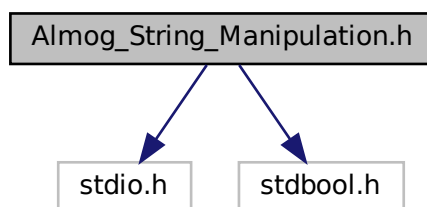
File Documentation

2.1 Almog_String_Manipulation.h File Reference

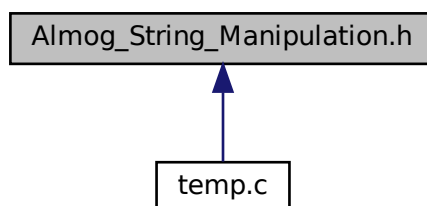
Lightweight string and line manipulation helpers.

```
#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_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_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

<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 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

<i>expr</i>	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

<i>expr</i>	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

<i>expr</i>	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

<i>expr</i>	An expression that yields a <code>size_t</code> . 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

<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 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

<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 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

<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 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

<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 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

<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 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

<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 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

<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 <code>ASM_MAX_LEN</code> 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

<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 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

<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 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

<i>c</i>	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

<i>c</i>	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

<i>c</i>	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

<i>c</i>	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

<code>c</code>	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

<code>c</code>	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

<code>c</code>	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

<code>c</code>	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

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

Returns

true if `c` is one of ' ',
'\t', '\n', '\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

<code>c</code>	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

<code>c</code>	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

<i>c</i>	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

<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 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

<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 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

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

<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 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

<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 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

<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 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

<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 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

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

Returns

The number of (possibly overlapping) occurrences found.

Note

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

Definition at line 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

<i>s</i>	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_from_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

<i>s1</i>	First string (may be shorter than <i>N</i>).
<i>s2</i>	Second string (may be shorter than <i>N</i>).
<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 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

```
00001
00039 #ifndef ALMOG_STRING_MANIPULATION_H_
00040 #define ALMOG_STRING_MANIPULATION_H_
00041
00042 #include <stdio.h>
00043 #include <stdbool.h>
00044
00060 #define ASM_MAX_LEN (int)1e3
00061
00069 #define asm_dprintSTRING(expr) printf(#expr " = %s\n", expr)
00070
00078 #define asm_dprintCHAR(expr) printf(#expr " = %c\n", expr)
00079
00087 #define asm_dprintINT(expr) printf(#expr " = %d\n", expr)
00088
00096 #define asm_dprintFLOAT(expr) printf(#expr " = %#g\n", expr)
00097
00105 #define asm_dprintDOUBLE(expr) printf(#expr " = %g\n", expr)
00106
00114 #define asm_dprintSIZE_T(expr) printf(#expr " = %zu\n", expr)
00115
00126 #define asm_min(a, b) ((a) < (b) ? (a) : (b))
00127
00138 #define asm_max(a, b) ((a) > (b) ? (a) : (b))
00139
00140 bool    asm_check_char_belong_to_base(char c, size_t base);
```

```

00141 void    asm_copy_array_by_indexes(char *target, int start, int end, char *src);
00142 size_t   asm_get_char_value_in_base(char c);
00143 int      asm_get_line(FILE *fp, char *dst);
00144 int      asm_get_next_word_from_line(char *dst, char *src, char delimiter);
00145 int      asm_get_word_and_cut(char *dst, char *src, char delimiter, bool leave_delimiter);
00146 bool     asm_isalnum(char c);
00147 bool     asm_isalpha(char c);
00148 bool     asm_iscntrl(char c);
00149 bool     asm_isdigit(char c);
00150 bool     asm_isgraph(char c);
00151 bool     asm_islower(char c);
00152 bool     asm_isprint(char c);
00153 bool     asm_isspace(char c);
00154 bool     asm_isupper(char c);
00155 bool     asm_isxdigit(char c);
00156 bool     asm_isXdigit(char c);
00157 void     asm_left_pad(char *s, size_t padding);
00158 size_t   asm_length(char *str);
00159 void     asm_remove_char_form_string(char *s, size_t index);
00160 int      asm_str_in_str(char *src, char *word_to_search);
00161 double   asm_str2double(char *s, char **end, size_t base);
00162 float    asm_str2float(char *s, char **end, size_t base);
00163 int      asm_str2int(char *s, char **end, size_t base);
00164 size_t   asm_str2size_t(char *s, char **end, size_t base);
00165 void     asm_strip_whitespace(char *s);
00166 int      asm_strncmp(const char *s1, const char *s2, const int N);
00167 void     asm_tolower(char *s);
00168 void     asm_toupper(char *s);
00169
00170
00171 #endif /*ALMOG_STRING_MANIPULATION_H_*/
00172
00173 #ifdef ALMOG_STRING_MANIPULATION_IMPLEMENTATION
00174 #undef ALMOG_STRING_MANIPULATION_IMPLEMENTATION
00175
00186 bool asm_check_char_belong_to_base(char c, size_t base)
00187 {
00188     if (base > 36 || base < 2) {
00189         fprintf(stderr, "%s:%d:\n%s:\n[Error] Supported bases are [2...36]. Inputted: %zu\n\n",
00190             __FILE__, __LINE__, __func__, base);
00191         return false;
00192     }
00193     if (base <= 10) {
00194         return c >= '0' && c <= '9' + (char)base - 10;
00195     }
00196     if (base > 10) {
00197         return asm_isdigit(c) || (c >= 'A' && c <= ('A' + (char)base - 11)) || (c >= 'a' && c <=
00198             ('a' + (char)base - 11));
00199     }
00200     return false;
00201 }
00202
00219 void asm_copy_array_by_indexes(char *target, int start, int end, char *src)
00220 {
00221     int j = 0;
00222     for (int i = start; i < end; i++) {
00223         target[j] = src[i];
00224         j++;
00225     }
00226     target[j] = '\0';
00227 }
00228
00238 size_t asm_get_char_value_in_base(char c)
00239 {
00240     if (asm_isdigit(c)) {
00241         return c - '0';
00242     } else if (asm_isupper(c)) {
00243         return c - 'A' + 10;
00244     } else {
00245         return c - 'a' + 10;
00246     }
00247 }
00248
00269 int asm_get_line(FILE *fp, char *dst)
00270 {
00271     int i = 0;
00272     int c;
00273
00274     while ((c = fgetc(fp)) != '\n' && c != EOF) {
00275         dst[i] = c;
00276         i++;
00277         if (i >= ASM_MAX_LEN) {
00278             fprintf(stderr, "%s:%d:\n%s:\n[Error] index exceeds ASM_MAX_LEN. Line in file is too
00279 long.\n\n", __FILE__, __LINE__, __func__);
00280             return -1;
00281         }
00282     }

```

```

00281     }
00282     dst[i] = '\0';
00283     if (c == EOF && i == 0) {
00284         return -1;
00285     }
00286     return i;
00287 }
00288
00318 int asm_get_next_word_from_line(char *dst, char *src, char delimiter)
00319 {
00320     int i = 0, j = 0;
00321     char c;
00322
00323     while (asm_isspace((c = src[i]))) {
00324         i++;
00325     }
00326
00327     while ((c = src[i]) != delimiter && c != '\n' && c != '\0') {
00328         dst[j] = src[i];
00329         i++;
00330         j++;
00331     }
00332
00333     if ((c == delimiter || c == '\n' || c == '\0') && i == 0) {
00334         dst[j++] = c;
00335         i++;
00336     }
00337
00338     dst[j] = '\0';
00339
00340     if (j == 0) {
00341         return -1;
00342     }
00343     return i;
00344 }
00345
00374 int asm_get_word_and_cut(char *dst, char *src, char delimiter, bool leave_delimiter)
00375 {
00376     int last_pos;
00377
00378     if (src[0] == '\0') {
00379         return 0;
00380     }
00381     last_pos = asm_get_next_word_from_line(dst, src, delimiter);
00382     if (last_pos == -1) {
00383         return 0;
00384     }
00385     if (leave_delimiter) {
00386         asm_copy_array_by_indexes(src, last_pos, asm_length(src), src);
00387     } else {
00388         asm_copy_array_by_indexes(src, last_pos + 1, asm_length(src), src);
00389     }
00390     return 1;
00391 }
00392
00399 bool asm_isalnum(char c)
00400 {
00401     return asm_isalpha(c) || asm_isdigit(c);
00402 }
00403
00410 bool asm_isalpha(char c)
00411 {
00412     return asm_isupper(c) || asm_islower(c);
00413 }
00414
00421 bool asm_iscntrl(char c)
00422 {
00423     if ((c >= 0 && c <= 31) || c == 127) {
00424         return true;
00425     } else {
00426         return false;
00427     }
00428 }
00429
00436 bool asm_isdigit(char c)
00437 {
00438     if (c >= '0' && c <= '9') {
00439         return true;
00440     } else {
00441         return false;
00442     }
00443 }
00444
00451 bool asm_isgraph(char c)
00452 {
00453     if (c >= 33 && c <= 126) {
00454         return true;

```

```

00455     } else {
00456         return false;
00457     }
00458 }
00459
00466 bool asm_islower(char c)
00467 {
00468     if (c >= 'a' && c <= 'z') {
00469         return true;
00470     } else {
00471         return false;
00472     }
00473 }
00474
00482 bool asm_isprint(char c)
00483 {
00484     return asm_isgraph(c) || c == ' ';
00485 }
00486
00494 bool asm_isspace(char c)
00495 {
00496     if ((c >= 33 && c <= 47) || (c >= 58 && c <= 64) || (c >= 91 && c <= 96) || (c >= 123 && c <=
00497         126)) {
00498         return true;
00499     } else {
00500         return false;
00501     }
00502 }
00510 bool asm_isspace(char c)
00511 {
00512     if (c == ' ' || c == '\n' || c == '\t' ||
00513         c == '\v' || c == '\f' || c == '\r') {
00514         return true;
00515     } else {
00516         return false;
00517     }
00518 }
00519
00526 bool asm_isupper(char c)
00527 {
00528     if (c >= 'A' && c <= 'Z') {
00529         return true;
00530     } else {
00531         return false;
00532     }
00533 }
00534
00541 bool asm_isxdigit(char c)
00542 {
00543     if ((c >= 'a' && c <= 'f') || asm_isdigit(c)) {
00544         return true;
00545     } else {
00546         return false;
00547     }
00548 }
00549
00556 bool asm_isXdigit(char c)
00557 {
00558     if ((c >= 'A' && c <= 'F') || asm_isdigit(c)) {
00559         return true;
00560     } else {
00561         return false;
00562     }
00563 }
00564
00578 void asm_left_pad(char *s, size_t padding)
00579 {
00580     int len = (int)asm_length(s);
00581     for (int i = len+1; i >= 0; i--) {
00582         s[i+(int)padding] = s[i];
00583     }
00584     for (int i = 0; i < (int)padding; i++) {
00585         s[i] = ' ';
00586     }
00587 }
00588
00599 size_t asm_length(char *str)
00600 {
00601     char c;
00602     size_t i = 0;
00603
00604     while ((c = str[i++]) != '\0') {
00605         if (i > ASM_MAX_LEN) {
00606             fprintf(stderr, "%s:%d:\n%s:\n[Error] index exceeds ASM_MAX_LEN_LINE. Probably no NULL
termination.\n\n", __FILE__, __LINE__, __func__);
00607             return __SIZE_MAX__;

```

```

00608     }
00609 }
00610 return --i;
00611 }
00612
00625 void asm_remove_char_form_string(char *s, size_t index)
00626 {
00627     size_t len = asm_length(s);
00628     if (len == 0) return;
00629     if (index >= len) {
00630         fprintf(stderr, "%s:%d:\n%s:\n[Error] index exceeds array length.\n\n", __FILE__, __LINE__,
__func__);
00631         return;
00632     }
00633
00634     for (size_t i = index; i < len; i++) {
00635         s[i] = s[i+1];
00636     }
00637 }
00638
00653 int asm_str_in_str(char *src, char *word_to_search)
00654 {
00655     int i = 0, num_of_accur = 0;
00656     while (src[i] != '\0') {
00657         if (asm_strncmp(src+i, word_to_search, asm_length(word_to_search))) {
00658             num_of_accur++;
00659         }
00660         i++;
00661     }
00662     return num_of_accur;
00663 }
00664
00683 double asm_str2double(char *s, char **end, size_t base)
00684 {
00685     if (base < 2 || base > 36) {
00686         fprintf(stderr, "%s:%d:\n%s:\n[Error] Supported bases are [2...36]. Input: %zu\n\n", __FILE__,
__LINE__, __func__, base);
00687         if (end) *end = s;
00688         return 0.0f;
00689     }
00690     while (asm_isspace(*s)) {
00691         s++;
00692     }
00693
00694     int i = 0;
00695     if (s[0] == '-' || s[0] == '+') {
00696         i++;
00697     }
00698     int sign = s[0] == '-' ? -1 : 1;
00699
00700     size_t left = 0;
00701     for (; asm_check_char_belong_to_base(s[i], base); i++) {
00702         left = base * left + asm_get_char_value_in_base(s[i]);
00703     }
00704     if (s[i] != '.') {
00705         if (end) *end = s + i;
00706         return (left * sign);
00707     }
00708
00709     i++; /* skip the point */
00710
00711     double right = 0;
00712     size_t divider = base;
00713     for (; asm_check_char_belong_to_base(s[i], base); i++) {
00714         right = right + asm_get_char_value_in_base(s[i]) / (double)divider;
00715         divider *= base;
00716     }
00717
00718     if (end) *end = s + i;
00719
00720     return sign * (left + right);
00721 }
00722
00741 float asm_str2float(char *s, char **end, size_t base)
00742 {
00743     if (base < 2 || base > 36) {
00744         fprintf(stderr, "%s:%d:\n%s:\n[Error] Supported bases are [2...36]. Input: %zu\n\n", __FILE__,
__LINE__, __func__, base);
00745         if (end) *end = s;
00746         return 0.0f;
00747     }
00748     while (asm_isspace(*s)) {
00749         s++;
00750     }
00751
00752     int i = 0;
00753     if (s[0] == '-' || s[0] == '+') {

```

```

00754         i++;
00755     }
00756     int sign = s[0] == '-' ? -1 : 1;
00757
00758     int left = 0;
00759     for (; asm_check_char_belong_to_base(s[i], base); i++) {
00760         left = base * left + asm_get_char_value_in_base(s[i]);
00761     }
00762     if (s[i] != '.') {
00763         if (end) *end = s + i;
00764         return left * sign;
00765     }
00766
00767     i++; /* skip the point */
00768
00769     float right = 0;
00770     size_t divider = base;
00771     for (; asm_check_char_belong_to_base(s[i], base); i++) {
00772         right = right + asm_get_char_value_in_base(s[i]) / (float)divider;
00773         divider *= base;
00774     }
00775
00776     if (end) *end = s + i;
00777
00778     return sign * (left + right);
00779 }
00780
00797 int asm_str2int(char *s, char **end, size_t base)
00798 {
00799     if (base < 2 || base > 36) {
00800         fprintf(stderr, "%s:%d:\n%s:\n[Error] Supported bases are [2...36]. Input: %zu\n\n", __FILE__,
00801             __LINE__, __func__, base);
00802         if (end) *end = s;
00803         return 0;
00804     }
00805     while (asm_isspace(*s)) {
00806         s++;
00807     }
00808
00809     int n = 0, i = 0;
00810     if (s[0] == '-' || s[0] == '+') {
00811         i++;
00812     }
00813     int sign = s[0] == '-' ? -1 : 1;
00814
00815     for (; asm_check_char_belong_to_base(s[i], base); i++) {
00816         n = base * n + asm_get_char_value_in_base(s[i]);
00817     }
00818
00819     if (end) *end = s + i;
00820
00821     return n * sign;
00822 }
00839 size_t asm_str2size_t(char *s, char **end, size_t base)
00840 {
00841     if (base < 2 || base > 36) {
00842         fprintf(stderr, "%s:%d:\n%s:\n[Error] Supported bases are [2...36]. Input: %zu\n\n", __FILE__,
00843             __LINE__, __func__, base);
00844         if (end) *end = s;
00845         return 0;
00846     }
00847     while (asm_isspace(*s)) {
00848         s++;
00849     }
00850
00851     if (s[0] == '-') {
00852         fprintf(stderr, "%s:%d:\n%s:\n[Error] Unable to convert a negative number to size_t.\n\n",
00853             __FILE__, __LINE__, __func__);
00854         if (end) *end = s;
00855         return 0;
00856     }
00857
00858     size_t n = 0, i = 0;
00859     if (s[0] == '+') {
00860         i++;
00861     }
00862
00863     for (; asm_check_char_belong_to_base(s[i], base); i++) {
00864         n = base * n + asm_get_char_value_in_base(s[i]);
00865     }
00866
00867     if (end) *end = s + i;
00868
00869     return n;
00870 }

```

```

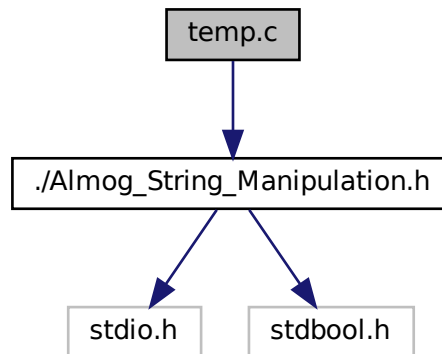
00879 void asm_strip_whitespace(char *s)
00880 {
00881     size_t len = asm_length(s);
00882     size_t i;
00883     for (i = 0; i < len; i++) {
00884         if (asm_isspace(s[i])) {
00885             asm_remove_char_from_string(s, i);
00886             len--;
00887             i--;
00888         }
00889     }
00890     s[i] = '\0';
00891 }
00892
00909 int asm_strncmp(const char *s1, const char *s2, const int N)
00910 {
00911     int i = 0;
00912     while (i < N) {
00913         if (s1[i] == '\0' && s2[i] == '\0') {
00914             break;
00915         }
00916         if (s1[i] != s2[i] || (s1[i] == '\0') || (s2[i] == '\0')) {
00917             return 0;
00918         }
00919         i++;
00920     }
00921     return 1;
00922 }
00923
00929 void asm_tolower(char *s)
00930 {
00931     size_t len = asm_length(s);
00932     for (size_t i = 0; i < len; i++) {
00933         if (asm_isupper(s[i])) {
00934             s[i] += 'a' - 'A';
00935         }
00936     }
00937 }
00938
00944 void asm_toupper(char *s)
00945 {
00946     size_t len = asm_length(s);
00947     for (size_t i = 0; i < len; i++) {
00948         if (asm_islower(s[i])) {
00949             s[i] += 'A' - 'a';
00950         }
00951     }
00952 }
00953
00954 #endif /*ALMOG_STRING_MANIPULATION_IMPLEMENTATION*/
00955

```

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

```
00001 #define ALMOG_STRING_MANIPULATION_IMPLEMENTATION
00002 #include "./Almog_String_Manipulation.h"
00003
00004 int main(void)
00005 {
00006     char str1[] = "10110110110110110110.101";
00007     char *str = str1;
00008     char *temp;
00009
00010     asm_dprintINT(asm_str2int(str, &temp, 10));
00011     asm_dprintSTRING(temp);
00012     asm_dprintSIZE_T(asm_str2size_t(str, &temp, 10));
00013     asm_dprintSTRING(temp);
00014     asm_dprintFLOAT(asm_str2float(str, &temp, 10));
00015     asm_dprintSTRING(temp);
00016     asm_dprintDOUBLE(asm_str2double(str, &temp, 10));
00017     asm_dprintSTRING(temp);
00018
00019     return 0;
00020 }
```


Index

Almog_String_Manipulation.h, [3](#)
asm_check_char_belong_to_base, [9](#)
asm_copy_array_by_indexes, [10](#)
asm_dprintCHAR, [6](#)
asm_dprintDOUBLE, [6](#)
asm_dprintFLOAT, [7](#)
asm_dprintINT, [7](#)
asm_dprintSIZE_T, [7](#)
asm_dprintSTRING, [8](#)
asm_get_char_value_in_base, [10](#)
asm_get_line, [11](#)
asm_get_next_word_from_line, [12](#)
asm_get_word_and_cut, [13](#)
asm_isalnum, [14](#)
asm_isalpha, [14](#)
asm_iscntrl, [14](#)
asm_isdigit, [15](#)
asm_isgraph, [15](#)
asm_islower, [16](#)
asm_isprint, [16](#)
asm_ispunct, [17](#)
asm_isspace, [17](#)
asm_isupper, [17](#)
asm_isXdigit, [18](#)
asm_isxdigit, [18](#)
asm_left_pad, [19](#)
asm_length, [19](#)
asm_max, [8](#)
ASM_MAX_LEN, [8](#)
asm_min, [9](#)
asm_remove_char_form_string, [20](#)
asm_str2double, [20](#)
asm_str2float, [21](#)
asm_str2int, [22](#)
asm_str2size_t, [22](#)
asm_str_in_str, [23](#)
asm_strip_whitespace, [23](#)
asm_strncmp, [24](#)
asm_tolower, [24](#)
asm_toupper, [25](#)
ALMOG_STRING_MANIPULATION_IMPLEMENTATION
temp.c, [32](#)
asm_check_char_belong_to_base
Almog_String_Manipulation.h, [9](#)
asm_copy_array_by_indexes
Almog_String_Manipulation.h, [10](#)
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, [8](#)
asm_get_char_value_in_base
Almog_String_Manipulation.h, [10](#)
asm_get_line
Almog_String_Manipulation.h, [11](#)
asm_get_next_word_from_line
Almog_String_Manipulation.h, [12](#)
asm_get_word_and_cut
Almog_String_Manipulation.h, [13](#)
asm_isalnum
Almog_String_Manipulation.h, [14](#)
asm_isalpha
Almog_String_Manipulation.h, [14](#)
asm_iscntrl
Almog_String_Manipulation.h, [14](#)
asm_isdigit
Almog_String_Manipulation.h, [15](#)
asm_isgraph
Almog_String_Manipulation.h, [15](#)
asm_islower
Almog_String_Manipulation.h, [16](#)
asm_isprint
Almog_String_Manipulation.h, [16](#)
asm_ispunct
Almog_String_Manipulation.h, [17](#)
asm_isspace
Almog_String_Manipulation.h, [17](#)
asm_isupper
Almog_String_Manipulation.h, [17](#)
asm_isXdigit
Almog_String_Manipulation.h, [18](#)
asm_isxdigit
Almog_String_Manipulation.h, [18](#)
asm_left_pad
Almog_String_Manipulation.h, [19](#)
asm_length
Almog_String_Manipulation.h, [19](#)
asm_max
Almog_String_Manipulation.h, [8](#)
ASM_MAX_LEN
Almog_String_Manipulation.h, [8](#)

- asm_min
 - Almog_String_Manipulation.h, [9](#)
- asm_remove_char_form_string
 - Almog_String_Manipulation.h, [20](#)
- asm_str2double
 - Almog_String_Manipulation.h, [20](#)
- asm_str2float
 - Almog_String_Manipulation.h, [21](#)
- asm_str2int
 - Almog_String_Manipulation.h, [22](#)
- asm_str2size_t
 - Almog_String_Manipulation.h, [22](#)
- asm_str_in_str
 - Almog_String_Manipulation.h, [23](#)
- asm_strip_whitespace
 - Almog_String_Manipulation.h, [23](#)
- asm_strncmp
 - Almog_String_Manipulation.h, [24](#)
- asm_tolower
 - Almog_String_Manipulation.h, [24](#)
- asm_toupper
 - Almog_String_Manipulation.h, [25](#)
- main
 - temp.c, [32](#)
- temp.c, [31](#)
 - ALMOG_STRING_MANIPULATION_IMPLEMENTATION,
[32](#)
 - main, [32](#)