

# Almog String Manipulation

Generated by Doxygen 1.9.1



---

<b>1 File Index</b>	<b>1</b>
1.1 File List . . . . .	1
<b>2 File Documentation</b>	<b>3</b>
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 <code>asm_dprintCHAR</code> . . . . .	6
2.1.2.2 <code>asm_dprintDOUBLE</code> . . . . .	6
2.1.2.3 <code>asm_dprintFLOAT</code> . . . . .	7
2.1.2.4 <code>asm_dprintINT</code> . . . . .	7
2.1.2.5 <code>asm_dprintSIZE_T</code> . . . . .	7
2.1.2.6 <code>asm_dprintSTRING</code> . . . . .	8
2.1.2.7 <code>asm_max</code> . . . . .	8
2.1.2.8 <code>ASM_MAX_LEN</code> . . . . .	9
2.1.2.9 <code>asm_min</code> . . . . .	9
2.1.3 Function Documentation . . . . .	9
2.1.3.1 <code>asm_check_char_belong_to_base()</code> . . . . .	9
2.1.3.2 <code>asm_copy_array_by_indexes()</code> . . . . .	10
2.1.3.3 <code>asm_get_char_value_in_base()</code> . . . . .	11
2.1.3.4 <code>asm_get_line()</code> . . . . .	11
2.1.3.5 <code>asm_get_next_word_from_line()</code> . . . . .	12
2.1.3.6 <code>asm_get_word_and_cut()</code> . . . . .	13
2.1.3.7 <code>asm_isalnum()</code> . . . . .	14
2.1.3.8 <code>asm_isalpha()</code> . . . . .	14
2.1.3.9 <code>asm_iscntrl()</code> . . . . .	15
2.1.3.10 <code>asm_isdigit()</code> . . . . .	15
2.1.3.11 <code>asm_isgraph()</code> . . . . .	15
2.1.3.12 <code>asm_islower()</code> . . . . .	16
2.1.3.13 <code>asm_isprint()</code> . . . . .	16
2.1.3.14 <code>asm_ispunct()</code> . . . . .	17
2.1.3.15 <code>asm_isspace()</code> . . . . .	17
2.1.3.16 <code>asm_isupper()</code> . . . . .	18
2.1.3.17 <code>asm_isxdigit()</code> . . . . .	18
2.1.3.18 <code>asm_isXdigit()</code> . . . . .	18
2.1.3.19 <code>asm_left_pad()</code> . . . . .	19
2.1.3.20 <code>asm_length()</code> . . . . .	19
2.1.3.21 <code>asm_remove_char_form_string()</code> . . . . .	20
2.1.3.22 <code>asm_str2double()</code> . . . . .	20
2.1.3.23 <code>asm_str2float()</code> . . . . .	21
2.1.3.24 <code>asm_str2int()</code> . . . . .	22
2.1.3.25 <code>asm_str2size_t()</code> . . . . .	22

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

# Chapter 1

## File Index

### 1.1 File List

Here is a list of all files with brief descriptions:

<a href="#">Almog_String_Manipulation.h</a>	Lightweight string and line manipulation helpers . . . . .	<a href="#">3</a>
<a href="#">temp.c</a>		<a href="#">31</a>



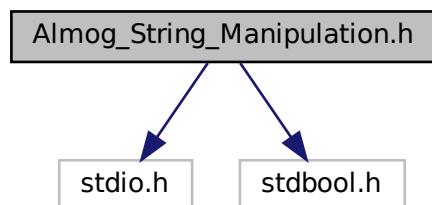
## Chapter 2

# File Documentation

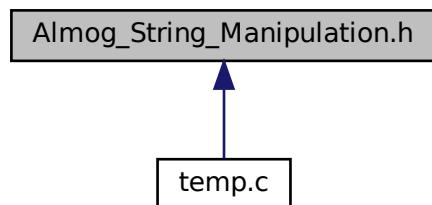
### 2.1 Almog\_String\_Manipulation.h File Reference

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)`

- `bool asm_isspace (char c)`  
*Test for a punctuation character (ASCII).*
- `bool asm_isupper (char c)`  
*Test for a whitespace character (ASCII).*
- `bool asm_isxdigit (char c)`  
*Test for an uppercase letter (ASCII).*
- `bool asm_isXdigit (char c)`  
*Test for a hexadecimal digit (lowercase or decimal).*
- `void asm_left_pad (char *s, size_t padding)`  
*Left-pad a string with spaces in-place.*
- `size_t asm_length (char *str)`  
*Compute the length of a null-terminated C string.*
- `void asm_remove_char_form_string (char *s, size_t index)`  
*Remove a single character from a string by index.*
- `int asm_str_in_str (char *src, char *word_to_search)`  
*Count occurrences of a substring within a string.*
- `double asm_str2double (char *s, char **end, size_t base)`  
*Convert a string to double in the given base.*
- `float asm_str2float (char *s, char **end, size_t base)`  
*Convert a string to float in the given base.*
- `int asm_str2int (char *s, char **end, size_t base)`  
*Convert a string to int in the given base.*
- `size_t asm_str2size_t (char *s, char **end, size_t base)`  
*Convert a string to size\_t in the given base.*
- `void asm_strip_whitespace (char *s)`  
*Remove all ASCII whitespace characters from a string in-place.*
- `int asm_strncmp (const char *s1, const char *s2, const int N)`  
*Compare up to N characters for equality (boolean result).*
- `void asm_tolower (char *s)`  
*Convert all ASCII letters in a string to lowercase in-place.*
- `void asm_toupper (char *s)`  
*Convert all ASCII letters in a string to uppercase in-place.*

## 2.1.1 Detailed Description

Lightweight string and line manipulation helpers.

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

- Reading a single line from a FILE stream
- Measuring string length
- Extracting the next "word" (token) from a line using a delimiter
- Cutting the extracted word from the source buffer
- Copying a substring by indices
- Counting occurrences of a substring
- A boolean-style strncmp (returns 1 on equality, 0 otherwise)

- ASCII-only character classification helpers (isalnum, isalpha, ...)
- ASCII case conversion (toupper / tolower)
- In-place whitespace stripping and left padding
- Base-N string-to-number conversion for int, size\_t, float, and double

## Usage

- In exactly one translation unit, define ALMOG\_STRING\_MANIPULATION\_IMPLEMENTATION before including this header to compile the implementation.
- In all other files, include the header without the macro to get declarations only.

## Notes and limitations

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

Definition in file [Almog\\_String\\_Manipulation.h](#).

## 2.1.2 Macro Definition Documentation

### 2.1.2.1 asm\_dprintCHAR

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

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

#### Parameters

<code>expr</code>	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**

<code>expr</code>	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**

<code>expr</code>	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**

<code>expr</code>	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 size_t. The expression is evaluated exactly once.
-------------	---

Definition at line 114 of file [Almog\\_String\\_Manipulation.h](#).

**2.1.2.6 asm\_dprintSTRING**

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

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

**Parameters**

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

Definition at line 69 of file [Almog\\_String\\_Manipulation.h](#).

**2.1.2.7 asm\_max**

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

Return the larger of two values (macro).

**Parameters**

<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

<code>a</code>	First value.
<code>b</code>	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  $i = \text{start}, \text{start} + 1, \dots, \text{end} - 1$  from *src* into *target*, then writes a terminating '\0'.

**Parameters**

<i>target</i>	Destination buffer. Must be large enough to hold $(\text{end} - \text{start})$ 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 $\text{end} \geq \text{start}$ ).
<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  $\text{target} == \text{src}$  and  $\text{start} > 0$  (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

<code>c</code>	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 ('\n') or EOF is encountered. The newline, if present, is not copied. The result is always null-terminated on normal (non-error) completion.

#### Parameters

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

#### Returns

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

## Return values

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

## Note

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

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

Definition at line 269 of file [Almog\\_String\\_Manipulation.h](#).

References [ASM\\_MAX\\_LEN](#).

### 2.1.3.5 `asm_get_next_word_from_line()`

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

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

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

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

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

Special case:

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

## Parameters

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

<code>dst</code>	Destination buffer for the extracted word (large enough for the token and terminating null).
<code>src</code>	Source buffer. Modified in-place if a word is found.
<code>delimiter</code>	Delimiter character to stop at.
<code>leave_delimiter</code>	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

<code>c</code>	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

<code>c</code>	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

<code>c</code>	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

<code>c</code>	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 ' ', '\n', '\t', '\v', '\f', or '\r'; false otherwise.

Definition at line 510 of file [Almog\\_String\\_Manipulation.h](#).

Referenced by [asm\\_get\\_next\\_word\\_from\\_line\(\)](#), [asm\\_str2double\(\)](#), [asm\\_str2float\(\)](#), [asm\\_str2int\(\)](#), [asm\\_str2size\\_t\(\)](#), and [asm\\_strip\\_whitespace\(\)](#).

### 2.1.3.16 `asm_isupper()`

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

Test for an uppercase letter (ASCII).

#### Parameters

<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**

c	Character to test.
---	--------------------

**Returns**

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

Definition at line 556 of file [Almog\\_String\\_Manipulation.h](#).

References [asm\\_isdigit\(\)](#).

### 2.1.3.19 asm\_left\_pad()

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

Left-pad a string with spaces in-place.

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

**Parameters**

s	String to pad. Modified in-place.
padding	Number of leading spaces to insert.

**Warning**

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

Definition at line 578 of file [Almog\\_String\\_Manipulation.h](#).

References [asm\\_length\(\)](#).

### 2.1.3.20 asm\_length()

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

Compute the length of a null-terminated C string.

**Parameters**

str	Null-terminated string (must be non-NUL).
-----	---

**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, <i>*end</i> 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, <i>*end</i> 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

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

Definition at line 879 of file [Almog\\_String\\_Manipulation.h](#).

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

### 2.1.3.28 `asm_strncmp()`

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

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

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

#### Parameters

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

#### Returns

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

#### Note

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

Definition at line 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_ispunct(char c);
00154 bool      asm_isspace(char c);
00155 bool      asm_isupper(char c);
00156 bool      asm_isxdigit(char c);
00157 bool      asm_isXdigit(char c);
00158 void      asm_left_pad(char *, size_t padding);
00159 size_t    asm_length(char *str);
00160 void      asm_remove_char_form_string(char *, size_t index);
00161 int       asm_str_in_str(char *src, char *word_to_search);
00162 double    asm_str2double(char *, char **end, size_t base);
00163 float     asm_str2float(char *, char **end, size_t base);
00164 int       asm_str2int(char *, char **end, size_t base);
00165 size_t    asm_str2size_t(char *, char **end, size_t base);
00166 void      asm_strip_whitespace(char *);
00167 int       asm_strncmp(const char *s1, const char *s2, const int N);
00168 void      asm_tolower(char *);
00169 void      asm_toupper(char *);

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 }
00201
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         }

```

```

00281     }
00282     dst[i] = '\0';
00283     if (c == EOF && i == 0) {
00284         return -1;
00285     }
00286     return i;
00287 }
00288
00289 int asm_get_next_word_from_line(char *dst, char *src, char delimiter)
00290 {
00291     int i = 0, j = 0;
00292     char c;
00293
00294     while (asm_isspace((c = src[i]))) {
00295         i++;
00296     }
00297
00298     while ((c = src[i]) != delimiter && c != '\n' && c != '\0') {
00299         dst[j] = src[i];
00300         i++;
00301         j++;
00302     }
00303
00304     if ((c == delimiter || c == '\n' || c == '\0') && i == 0) {
00305         dst[j++] = c;
00306         i++;
00307     }
00308
00309     dst[j] = '\0';
00310
00311     if (j == 0) {
00312         return -1;
00313     }
00314     return i;
00315 }
00316
00317 int asm_get_word_and_cut(char *dst, char *src, char delimiter, bool leave_delimiter)
00318 {
00319     int last_pos;
00320
00321     if (src[0] == '\0') {
00322         return 0;
00323     }
00324     last_pos = asm_get_next_word_from_line(dst, src, delimiter);
00325     if (last_pos == -1) {
00326         return 0;
00327     }
00328     if (leave_delimiter) {
00329         asm_copy_array_by_indexes(src, last_pos, asm_length(src), src);
00330     } else {
00331         asm_copy_array_by_indexes(src, last_pos + 1, asm_length(src), src);
00332     }
00333     return 1;
00334 }
00335
00336 bool asm_isalnum(char c)
00337 {
00338     return asm_isalpha(c) || asm_isdigit(c);
00339 }
00340
00341 bool asm_isalpha(char c)
00342 {
00343     return asm_isupper(c) || asm_islower(c);
00344 }
00345
00346 bool asm_iscntrl(char c)
00347 {
00348     if ((c >= 0 && c <= 31) || c == 127) {
00349         return true;
00350     } else {
00351         return false;
00352     }
00353 }
00354
00355 bool asm_isdigit(char c)
00356 {
00357     if (c >= '0' && c <= '9') {
00358         return true;
00359     } else {
00360         return false;
00361     }
00362 }
00363
00364 bool asm_isgraph(char c)
00365 {
00366     if (c >= 33 && c <= 126) {
00367         return true;
00368     }
00369 }
00370
00371 bool asm_isprint(char c)
00372 {
00373     if (c >= 32 && c <= 126) {
00374         return true;
00375     }
00376 }
00377
00378 bool asm_ispunct(char c)
00379 {
00380     if (c >= 33 && c <= 47) {
00381         return true;
00382     } else if (c >= 58 && c <= 64) {
00383         return true;
00384     } else if (c >= 91 && c <= 96) {
00385         return true;
00386     } else if (c >= 123 && c <= 126) {
00387         return true;
00388     }
00389     return false;
00390 }
00391 }
```

```

00455     } else {
00456         return false;
00457     }
00458 }
00459
00460 bool asm_islower(char c)
00461 {
00462     if (c >= 'a' && c <= 'z') {
00463         return true;
00464     } else {
00465         return false;
00466     }
00467 }
00468
00469 bool asm_isprint(char c)
00470 {
00471     return asm_isgraph(c) || c == ' ';
00472 }
00473
00474
00475 bool asm_ispunct(char c)
00476 {
00477     if ((c >= 33 && c <= 47) || (c >= 58 && c <= 64) || (c >= 91 && c <= 96) || (c >= 123 && c <=
00478 126)) {
00479         return true;
00480     } else {
00481         return false;
00482     }
00483 }
00484
00485
00486 bool asm_isspace(char c)
00487 {
00488     if (c == ' ' || c == '\n' || c == '\t' ||
00489      c == '\v' || c == '\f' || c == '\r') {
00490         return true;
00491     } else {
00492         return false;
00493     }
00494 }
00495
00496
00497 bool asm_isupper(char c)
00498 {
00499     if (c >= 'A' && c <= 'Z') {
00500         return true;
00501     } else {
00502         return false;
00503     }
00504 }
00505
00506
00507 bool asm_isxdigit(char c)
00508 {
00509     if ((c >= 'a' && c <= 'f') || asm_isdigit(c)) {
00510         return true;
00511     } else {
00512         return false;
00513     }
00514 }
00515
00516
00517 bool asm_isXdigit(char c)
00518 {
00519     if ((c >= 'A' && c <= 'F') || asm_isdigit(c)) {
00520         return true;
00521     } else {
00522         return false;
00523     }
00524 }
00525
00526
00527 void asm_left_pad(char *s, size_t padding)
00528 {
00529     int len = (int)asm_length(s);
00530     for (int i = len+1; i >= 0; i--) {
00531         s[i+(int)padding] = s[i];
00532     }
00533     for (int i = 0; i < (int)padding; i++) {
00534         s[i] = ' ';
00535     }
00536 }
00537
00538
00539 size_t asm_length(char *str)
00540 {
00541     char c;
00542     size_t i = 0;
00543
00544     while ((c = str[i++]) != '\0') {
00545         if (i > ASM_MAX_LEN) {
00546             fprintf(stderr, "%s:%d:%n%s:\n[%Error] index exceeds ASM_MAX_LEN_LINE. Probably no NULL
00547 termination.\n%n", __FILE__, __LINE__, __func__);
00548         }
00549     }
00550     return __SIZE_MAX__;
00551 }
```

```

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__,
00631             __func__);
00632         return;
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__,
00687             __LINE__, __func__, base);
00688         if (*end) *end = s;
00689         return 0.0f;
00690     }
00691     while (asm_isspace(*s)) {
00692         s++;
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__,
00745             __LINE__, __func__, base);
00746         if (*end) *end = s;
00747         return 0.0f;
00748     }
00749     while (asm_isspace(*s)) {
00750         s++;
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",
00801             __LINE__, __func__, base);
00802         if (*end) *end = s;
00803         return 0;
00804     }
00805     while (asm_isspace(*s)) {
00806         s++;
00807     }
00808     int n = 0, i = 0;
00809     if (s[0] == '-' || s[0] == '+') {
00810         i++;
00811     }
00812     int sign = s[0] == '-' ? -1 : 1;
00813
00814     for (; asm_check_char_belong_to_base(s[i], base); i++) {
00815         n = base * n + asm_get_char_value_in_base(s[i]);
00816     }
00817
00818     if (*end) *end = s + i;
00819
00820     return n * sign;
00821 }
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",
00843             __LINE__, __func__, base);
00844         if (*end) *end = s;
00845         return 0;
00846     }
00847     while (asm_isspace(*s)) {
00848         s++;
00849     }
00850     if (s[0] == '-') {
00851         fprintf(stderr, "%s:%d:\n%s:\n[Error] Unable to convert a negative number to size_t.\n\n",
00852             __FILE__, __LINE__, __func__);
00853         if (*end) *end = s;
00854         return 0;
00855     }
00856     size_t n = 0, i = 0;
00857     if (s[0] == '+') {
00858         i++;
00859     }
00860
00861     for (; asm_check_char_belong_to_base(s[i], base); i++) {
00862         n = base * n + asm_get_char_value_in_base(s[i]);
00863     }
00864
00865     if (*end) *end = s + i;
00866
00867     return n;
00868 }
00869

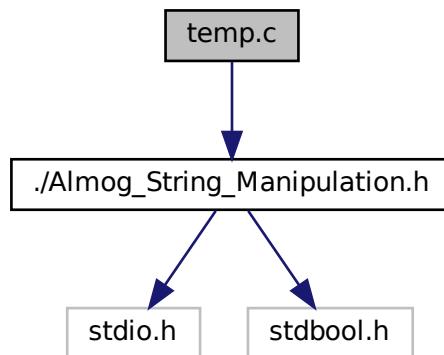
```

```
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(asn_str2int(str, &temp, 10));
00011     asm_dprintSTRING(temp);
00012     asm_dprintSIZE_T(asn_str2size_t(str, &temp, 10));
00013     asm_dprintSTRING(temp);
00014     asm_dprintFLOAT(asn_str2float(str, &temp, 10));
00015     asm_dprintSTRING(temp);
00016     asm_dprintDOUBLE(asn_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