

My library

Generated by Doxygen 1.9.1



<b>1 myLibrary homepage</b>	<b>1</b>
1.1 Hi!	1
<b>2 Data Structure Index</b>	<b>3</b>
2.1 Data Structures	3
<b>3 File Index</b>	<b>5</b>
3.1 File List	5
<b>4 Data Structure Documentation</b>	<b>7</b>
4.1 ArrayList Struct Reference	7
4.1.1 Detailed Description	7
4.1.2 Field Documentation	7
4.1.2.1 body	7
4.1.2.2 size	8
4.1.2.3 type	8
4.2 LinkedList Struct Reference	8
4.2.1 Detailed Description	9
4.2.2 Field Documentation	9
4.2.2.1 head	9
4.2.2.2 size	9
4.2.2.3 tail	9
4.2.2.4 type	9
4.3 node Struct Reference	10
4.3.1 Detailed Description	10
4.3.2 Field Documentation	10
4.3.2.1 data	10
4.3.2.2 linked	10
<b>5 File Documentation</b>	<b>11</b>
5.1 arrayList.h File Reference	11
5.1.1 Macro Definition Documentation	13
5.1.1.1 newALFromArray	13
5.1.2 Function Documentation	13
5.1.2.1 appendToAL()	13
5.1.2.2 areALEqual()	14
5.1.2.3 bubbleSortAL()	14
5.1.2.4 chooseNewALFromArray()	15
5.1.2.5 deleteAL()	15
5.1.2.6 getFromAL()	15
5.1.2.7 insertToAL()	16
5.1.2.8 isInAL()	16
5.1.2.9 linearSearchAL()	17
5.1.2.10 mergeAL()	17

5.1.2.11 newAL()	18
5.1.2.12 newALFromAL()	18
5.1.2.13 newALFromByteArray()	18
5.1.2.14 newALFromCharArray()	18
5.1.2.15 newALFromDoubleArray()	19
5.1.2.16 newALFromFloatArray()	19
5.1.2.17 newALFromIntArray()	19
5.1.2.18 newALFromPtrArray()	19
5.1.2.19 printAL()	19
5.1.2.20 quickSortAL()	20
5.1.2.21 removeFromAL()	20
5.1.2.22 reverseAL()	20
5.1.2.23 setALItem()	21
5.1.2.24 sliceAL()	21
5.2 arrays.h File Reference	21
5.2.1 Detailed Description	23
5.2.2 Function Documentation	23
5.2.2.1 charBubbleSort()	23
5.2.2.2 charQuickSort()	23
5.2.2.3 chooseBubbleSortArr()	23
5.2.2.4 chooseLinearSearch()	24
5.2.2.5 chooseQuickSortArr()	24
5.2.2.6 doubleBubbleSort()	25
5.2.2.7 doubleQuickSort()	25
5.2.2.8 floatBubbleSort()	25
5.2.2.9 floatQuickSort()	25
5.2.2.10 intBubbleSort()	26
5.2.2.11 intQuickSort()	26
5.2.2.12 printMatrix()	26
5.2.2.13 ptrBubbleSort()	27
5.2.2.14 ptrQuickSort()	27
5.3 constants.h File Reference	27
5.3.1 Detailed Description	28
5.3.2 Macro Definition Documentation	28
5.3.2.1 EQUAL	28
5.3.2.2 FALSE	28
5.3.2.3 GREATER	28
5.3.2.4 KEY_NOT_FOUND	28
5.3.2.5 SMALLER	29
5.3.2.6 TRUE	29
5.4 linkedList.h File Reference	29
5.4.1 Function Documentation	30

5.4.1.1 appendToLL()	30
5.4.1.2 appendToLLFromPtr()	31
5.4.1.3 areLLEqual()	31
5.4.1.4 chooseNewLLFromArray()	32
5.4.1.5 deleteLL()	32
5.4.1.6 getFromLL()	32
5.4.1.7 insertToLL()	33
5.4.1.8 isInLL()	33
5.4.1.9 linearSearchLL()	34
5.4.1.10 linearSearchLLPtr()	34
5.4.1.11 mergeLL()	35
5.4.1.12 newLL()	35
5.4.1.13 newLLFromLL()	36
5.4.1.14 printLL()	36
5.4.1.15 removeFromLL()	36
5.4.1.16 setLLItem()	37
5.4.1.17 sliceLL()	37
5.5 macros.h File Reference	37
5.5.1 Detailed Description	38
5.5.2 Macro Definition Documentation	39
5.5.2.1 bubbleSortArr	39
5.5.2.2 cmpVal	39
5.5.2.3 quickSortArr	40
5.6 myLibrary.h File Reference	40
5.6.1 Detailed Description	41
5.7 strings.h File Reference	41
5.7.1 Detailed Description	42
5.7.2 Function Documentation	42
5.7.2.1 changeLastCharacter()	42
5.7.2.2 copyOf()	43
5.7.2.3 endsWith()	43
5.7.2.4 getLength()	44
5.7.2.5 getString()	44
5.8 types.h File Reference	44
5.8.1 Detailed Description	45
5.8.2 Typedef Documentation	45
5.8.2.1 byte	45
5.8.2.2 Node	45
5.8.2.3 spec_t	46
5.8.2.4 string	46
5.9 utility.h File Reference	46
5.9.1 Detailed Description	47

5.9.2 Function Documentation . . . . .	47
5.9.2.1 byteCmp() . . . . .	47
5.9.2.2 charCmp() . . . . .	47
5.9.2.3 chooseCmp() . . . . .	48
5.9.2.4 doubleCmp() . . . . .	48
5.9.2.5 floatCmp() . . . . .	48
5.9.2.6 intCmp() . . . . .	49
5.9.2.7 ptrCmp() . . . . .	49
5.9.2.8 saferMalloc() . . . . .	49
5.9.2.9 saferRealloc() . . . . .	49
<b>Index</b>	<b>51</b>

# Chapter 1

## myLibrary homepage

### 1.1 Hi!

Actually I don't know what I should put here, so at the moment I just suggest you to go to the [files](#) section. The source code and binaries are available [here](#). [Here](#) there is a PDF version of the docs.





## Chapter 2

# Data Structure Index

### 2.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">ArrayList</a>		
	<a href="#">ArrayList</a> type . . . . .	<a href="#">7</a>
<a href="#">LinkedList</a>		
	<a href="#">LinkedList</a> type . . . . .	<a href="#">8</a>
<a href="#">node</a>		
	<a href="#">Node</a> type . . . . .	<a href="#">10</a>



## Chapter 3

# File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

<a href="#">arrayList.h</a>	11
<a href="#">arrays.h</a>	
Common tasks with arrays: sorting, searching, printing etc	21
<a href="#">constants.h</a>	
Definition of symbolic constants used by the library	27
<a href="#">linkedList.h</a>	29
<a href="#">macros.h</a>	
Macros for emulated overloading	37
<a href="#">myLibrary.h</a>	
Includes all other headers. Useful for rapid import	40
<a href="#">strings.h</a>	
Common tasks with strings	41
<a href="#">types.h</a>	
Collection of useful types	44
<a href="#">utility.h</a>	
Common tasks such as comparing variables, swap bools, allocate memory	46



## Chapter 4

# Data Structure Documentation

### 4.1 ArrayList Struct Reference

[ArrayList](#) type

```
#include <types.h>
```

#### Data Fields

- [spec\\_t](#) type

*The type of the elements contained by the [ArrayList](#). Refer to [spec\\_t](#).*

- void \* [body](#)

*Void pointer to the first element of the [ArrayList](#).*

- unsigned int [size](#)

*The number of elements contained by the [ArrayList](#).*

#### 4.1.1 Detailed Description

[ArrayList](#) type

##### Note

All the parameters in this structure must be intended as read-only. Manually modifying them can cause unknown and unwanted behavior

#### 4.1.2 Field Documentation

##### 4.1.2.1 body

```
void* ArrayList::body
```

Void pointer to the first element of the [ArrayList](#).

#### 4.1.2.2 size

```
unsigned int ArrayList::size
```

The number of elements contained by the [ArrayList](#).

#### 4.1.2.3 type

```
spec_t ArrayList::type
```

The type of the elements contained by the [ArrayList](#). Refer to [spec\\_t](#).

The documentation for this struct was generated from the following file:

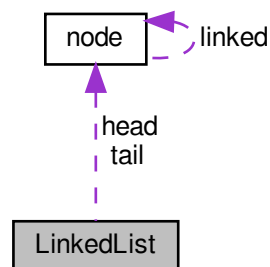
- [types.h](#)

## 4.2 LinkedList Struct Reference

[LinkedList](#) type

```
#include <types.h>
```

Collaboration diagram for [LinkedList](#):



### Data Fields

- [spec\\_t](#) type  
The type of the elements contained by the [LinkedList](#). Refer to [spec\\_t](#).
- [Node](#) head  
Head of the [LinkedList](#).
- [Node](#) tail  
Tail of the [LinkedList](#).
- unsigned int [size](#)  
The number of elements contained by the [LinkedList](#).

### 4.2.1 Detailed Description

[LinkedList](#) type

#### Note

All the parameters in this structure must be intended as read-only. Manually modifying them can cause unknown and unwanted behavior

### 4.2.2 Field Documentation

#### 4.2.2.1 head

[Node](#) [LinkedList::head](#)

Head of the [LinkedList](#).

#### 4.2.2.2 size

unsigned int [LinkedList::size](#)

The number of elements contained by the [LinkedList](#).

#### 4.2.2.3 tail

[Node](#) [LinkedList::tail](#)

Tail of the [LinkedList](#).

#### 4.2.2.4 type

[spec\\_t](#) [LinkedList::type](#)

The type of the elements contained by the [LinkedList](#). Refer to [spec\\_t](#).

The documentation for this struct was generated from the following file:

- [types.h](#)

## 4.3 node Struct Reference

Node type

```
#include <types.h>
```

Collaboration diagram for node:



### Data Fields

- void \* [data](#)  
*Pointer to the value contained.*
- struct [node](#) \* [linked](#)  
*The [Node](#) this [Node](#) is linked to.*

### 4.3.1 Detailed Description

Node type

Base component of every linked data type

Note

All the parameters in this structure must be intended as read-only. Manually modifying them can cause unknown and unwanted behavior

### 4.3.2 Field Documentation

#### 4.3.2.1 data

```
void* node::data
```

Pointer to the value contained.

#### 4.3.2.2 linked

```
struct node* node::linked
```

The [Node](#) this [Node](#) is linked to.

The documentation for this struct was generated from the following file:

- [types.h](#)



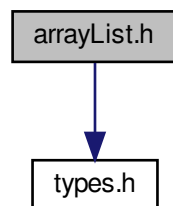
## Chapter 5

# File Documentation

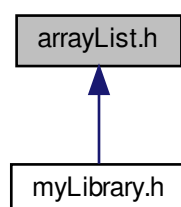
### 5.1 arrayList.h File Reference

```
#include "types.h"
```

Include dependency graph for arrayList.h:



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



## Macros

- `#define newALFromArray(list, size)`  
Create an *ArrayList* from a static list.

## Functions

- `ArrayList newAL (const spec_t spec)`  
Allocate a new *ArrayList* of specified type.
- `ArrayList newALFromAL (const ArrayList list)`  
Get a copy of an *ArrayList*.
- `void appendToAL (ArrayList list,...)`  
Insert an item at the end of an *ArrayList*.
- `void insertToAL (ArrayList list, unsigned int index,...)`  
Insert an element at a specified position of an *ArrayList*.
- `void setALItem (ArrayList list, unsigned int index,...)`  
Set value of an element of an *ArrayList*.
- `void mergeAL (ArrayList list1, const ArrayList list2)`  
Merge two *ArrayList*.
- `void sliceAL (ArrayList list, unsigned int begin, unsigned int end)`  
Slice an *ArrayList*.
- `void printAL (const spec_t spec, const ArrayList list)`  
Print contents from an *ArrayList*.
- `void removeFromAL (ArrayList list, unsigned int index)`  
Remove an item from an *ArrayList*.
- `void getFromAL (const ArrayList list, unsigned int index, void *dest)`  
Get an item from an *ArrayList*.
- `void deleteAL (ArrayList list)`  
Delete an *ArrayList*.
- `byte areALEqual (const ArrayList list1, const ArrayList list2)`  
Compare two *ArrayList*.
- `void reverseAL (ArrayList list)`  
Reverse an *ArrayList*.
- `void bubbleSortAL (ArrayList list)`  
Bubble sort for *ArrayList*.
- `void quickSortAL (ArrayList list)`  
Quicksort for *ArrayList*.
- `byte isInAL (ArrayList list,...)`  
Detect if an element is inside an *ArrayList*.
- `int linearSearchAL (ArrayList list,...)`  
Linear search for *ArrayList*.
- `ArrayList chooseNewALFromArray (const spec_t spec, const void *list, unsigned int size)`  
Create an *ArrayList* from an list.
- `ArrayList newALFromCharArray (const char list[], unsigned int size)`  
Create *ArrayList* from an list of chars.
- `ArrayList newALFromByteArray (const char list[], unsigned int size)`  
Alias for `newALFromCharArray()`. Used to create *ArrayList* from byte list. Refer to `newALFromCharArray()`
- `ArrayList newALFromIntArray (const int list[], unsigned int size)`  
Create *ArrayList* from an list of ints.
- `ArrayList newALFromFloatArray (const float list[], unsigned int size)`

Create [ArrayList](#) from an list of floats.

- [ArrayList](#) `newALFromDoubleArray` (const double list[], unsigned int size)

Create [ArrayList](#) from an list of doubles.

- [ArrayList](#) `newALFromPtrArray` (const void \*list, unsigned int size)

Create [ArrayList](#) from an list of pointers.

## 5.1.1 Macro Definition Documentation

### 5.1.1.1 `newALFromArray`

```
#define newALFromArray(  
    list,  
    size )
```

**Value:**

```
_Generic(list, char *  
: newALFromCharArray, int *  
: newALFromArray, float *  
: newALFromArray, double *  
: newALFromDoubleArray)(list, size)
```

Create an [ArrayList](#) from a static list.

**Parameters**

<i>list</i>	The list you want to create an <a href="#">ArrayList</a> from
<i>size</i>	The size of <code>list</code>

**Note**

Passing an list of pointers is not supported

**Returns**

An [ArrayList](#) containing all the elements of `list`

## 5.1.2 Function Documentation

### 5.1.2.1 `appendToAL()`

```
void appendToAL (  
    ArrayList list,  
    ... )
```

Insert an item at the end of an [ArrayList](#).

**Parameters**

<i>list</i>	The <a href="#">ArrayList</a> you want to append an item to
...	The item you want to append to <code>list</code>

**Note**

Even though appending more than one item does not throw a compiler nor runtime error, only appending one item is supported. Other items are ignored and are not appended to `list`. If you don't specify any item to be appended, still no errors occur but the content of your [ArrayList](#) can be messed up

**5.1.2.2 areALEqual()**

```
byte areALEqual (
    const ArrayList list1,
    const ArrayList list2 )
```

Compare two [ArrayList](#).

**Parameters**

<i>list1</i>	The first <a href="#">ArrayList</a> you want to compare
<i>list2</i>	The second <a href="#">ArrayList</a> you want to compare

**Returns**

The result of the comparison

**Return values**

<i>TRUE</i>	<code>list1</code> and <code>list2</code> have equal type, equal length and equal contents
<i>FALSE</i>	<code>list1</code> and <code>list2</code> do not have equal type, equal length or equal contents

**5.1.2.3 bubbleSortAL()**

```
void bubbleSortAL (
    ArrayList list )
```

Bubble sort for [ArrayList](#).

**Parameters**

<i>list</i>	The <a href="#">ArrayList</a> you want to bubble sort
-------------	---

#### 5.1.2.4 chooseNewALFromArray()

```
ArrayList chooseNewALFromArray (
    const spec_t spec,
    const void * list,
    unsigned int size )
```

Create an [ArrayList](#) from an list.

##### Parameters

<i>spec</i>	The type specifier of the list passed. Refer to <code>spec_t</code>
<i>list</i>	The list you want to create the <a href="#">ArrayList</a> from
<i>size</i>	The number of items of <code>list</code>

##### Returns

An [ArrayList](#) containing the elements in `list` in the same order

#### 5.1.2.5 deleteAL()

```
void deleteAL (
    ArrayList list )
```

Delete an [ArrayList](#).

##### Parameters

<i>list</i>	The <a href="#">ArrayList</a> you want to delete
-------------	--

#### 5.1.2.6 getFromAL()

```
void getFromAL (
    const ArrayList list,
    unsigned int index,
    void * dest )
```

Get an item from an [ArrayList](#).

##### Parameters

<i>list</i>	The <a href="#">ArrayList</a> you want to get an item from
<i>index</i>	The index of the item you want to get
<i>dest</i>	The address of the variable you want to store the item in

### 5.1.2.7 insertToAL()

```
void insertToAL (
    ArrayList list,
    unsigned int index,
    ... )
```

Insert an element at a specified position of an [ArrayList](#).

#### Parameters

<i>list</i>	The <a href="#">ArrayList</a> you want to insert an element into
<i>index</i>	The position you want to insert an item at
...	The item you want to insert into <code>list</code>

#### Note

Even though inserting more than one item does not throw a compiler nor runtime error, only inserting one item is supported. Other items are ignored and are not inserted into `list`. If you don't specify any item to be inserted, still no errors occur but the content of your [ArrayList](#) can be messed up

### 5.1.2.8 isInAL()

```
byte isInAL (
    ArrayList list,
    ... )
```

Detect if an element is inside an [ArrayList](#).

#### Parameters

<i>list</i>	The <a href="#">ArrayList</a> you want search in
...	The element you want to search

#### Note

Even though inserting zero more than one item does not throw a compiler nor runtime error, only searching one item is supported. Other items are ignored. If you don't specify any item to be searched, still no errors occur but the return value of the function can be unpredictable

#### Return values

<i>TRUE</i>	Given element is contained in <code>list</code>
<i>FALSE</i>	Given element is not contained in <code>list</code>

### 5.1.2.9 linearSearchAL()

```
int linearSearchAL (
    ArrayList list,
    ... )
```

Linear search for [ArrayList](#).

#### Parameters

<i>list</i>	The <a href="#">ArrayList</a> to be inspected
<i>...</i>	The key to be searched

#### Note

This function does not support float and double [ArrayList](#)

Even though passing more than one key does not throw a compiler nor runtime error, only searching one item is supported. Other items are ignored. If you don't specify any item to be searched, still no errors occur but the return value of the function can be unpredictable

#### Returns

The index of the first occurrence of the key in the list or the return code of the function

#### Return values

<i>KEY_NOT_FOUND</i>	The key was not found
----------------------	-----------------------

### 5.1.2.10 mergeAL()

```
void mergeAL (
    ArrayList list1,
    const ArrayList list2 )
```

Merge two [ArrayList](#).

#### Parameters

<i>list1</i>	The first <a href="#">ArrayList</a> to be merged, where the merged <a href="#">ArrayList</a> is saved
<i>list2</i>	The second <a href="#">ArrayList</a> to be merged

#### 5.1.2.11 newAL()

```
ArrayList newAL (
    const spec_t spec )
```

Allocate a new [ArrayList](#) of specified type.

##### Parameters

<i>spec</i>	Type specifier of the <a href="#">ArrayList</a> you want to create
-------------	--

##### Returns

An empty [ArrayList](#)

#### 5.1.2.12 newALFromAL()

```
ArrayList newALFromAL (
    const ArrayList list )
```

Get a copy of an [ArrayList](#).

##### Parameters

<i>list</i>	The <a href="#">ArrayList</a> you want to copy
-------------	--

##### Returns

A copy of `list`

#### 5.1.2.13 newALFromByteArray()

```
ArrayList newALFromByteArray (
    const char list[],
    unsigned int size )
```

Alias for [newALFromCharArray\(\)](#). Used to create [ArrayList](#) from byte list. Refer to [newALFromCharArray\(\)](#)

#### 5.1.2.14 newALFromCharArray()

```
ArrayList newALFromCharArray (
    const char list[],
    unsigned int size )
```

Create [ArrayList](#) from an list of chars.

Equivalent to `chooseNewALFromArray("%c", list, size)`. Refer to [chooseNewALFromArray\(\)](#)



#### 5.1.2.15 newALFromDoubleArray()

```
ArrayList newALFromDoubleArray (
    const double list[],
    unsigned int size )
```

Create [ArrayList](#) from an list of doubles.

Equivalent to `chooseNewALFromArray("%lf", list, size)`. Refer to [chooseNewALFromArray\(\)](#)

#### 5.1.2.16 newALFromFloatArray()

```
ArrayList newALFromFloatArray (
    const float list[],
    unsigned int size )
```

Create [ArrayList](#) from an list of floats.

Equivalent to `chooseNewALFromArray("%f", list, size)`. Refer to [chooseNewALFromArray\(\)](#)

#### 5.1.2.17 newALFromIntArray()

```
ArrayList newALFromIntArray (
    const int list[],
    unsigned int size )
```

Create [ArrayList](#) from an list of ints.

Equivalent to `chooseNewALFromArray("%i", list, size)`. Refer to [chooseNewALFromArray\(\)](#)

#### 5.1.2.18 newALFromPtrArray()

```
ArrayList newALFromPtrArray (
    const void * list,
    unsigned int size )
```

Create [ArrayList](#) from an list of pointers.

Equivalent to `chooseNewALFromArray("%p", list, size)`. Refer to [chooseNewALFromArray\(\)](#)

#### 5.1.2.19 printAL()

```
void printAL (
    const spec_t spec,
    const ArrayList list )
```

Print contents from an [ArrayList](#).

## Parameters

<i>spec</i>	The type and format specifier you want to use to print the single element of the <a href="#">ArrayList</a>
<i>list</i>	The <a href="#">ArrayList</a> you want to print

**5.1.2.20 quickSortAL()**

```
void quickSortAL (
    ArrayList list )
```

Quicksort for [ArrayList](#).

## Parameters

<i>list</i>	The <a href="#">ArrayList</a> you want to quicksort
-------------	---

**5.1.2.21 removeFromAL()**

```
void removeFromAL (
    ArrayList list,
    unsigned int index )
```

Remove an item from an [ArrayList](#).

## Parameters

<i>list</i>	The <a href="#">ArrayList</a> you want to delete an item from
<i>index</i>	The index of the item you want to delete

**5.1.2.22 reverseAL()**

```
void reverseAL (
    ArrayList list )
```

Reverse an [ArrayList](#).

## Parameters

<i>list</i>	The <a href="#">ArrayList</a> you want to reverse
-------------	---

### 5.1.2.23 setALItem()

```
void setALItem (
    ArrayList list,
    unsigned int index,
    ... )
```

Set value of an element of an [ArrayList](#).

#### Parameters

<i>list</i>	The <a href="#">ArrayList</a> you want to edit
<i>index</i>	The index of the element you want to change
<i>...</i>	The item you want to insert into <code>list</code>

#### Note

Even though inserting more than one item does not throw a compiler nor runtime error, only setting one item is supported. Other items are ignored. If you don't specify any item to be inserted, still no errors occur but the content of your [ArrayList](#) can be messed up

### 5.1.2.24 sliceAL()

```
void sliceAL (
    ArrayList list,
    unsigned int begin,
    unsigned int end )
```

Slice an [ArrayList](#).

#### Parameters

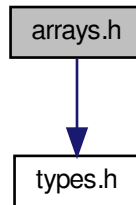
<i>list</i>	The <a href="#">ArrayList</a> you want to slice, where the sliced <a href="#">ArrayList</a> is saved
<i>begin</i>	The index of the beginning of the slice
<i>end</i>	The index of the end of the slice

## 5.2 arrays.h File Reference

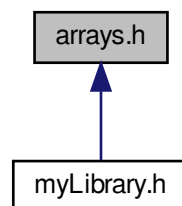
Common tasks with arrays: sorting, searching, printing etc.

```
#include "types.h"
```

Include dependency graph for arrays.h:



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



## Functions

- void [chooseBubbleSortArr](#) (const [spec\\_t](#) spec, void \*arr, unsigned int size)  
*Bubble sort for arrays.*
- void [chooseQuickSortArr](#) (const [spec\\_t](#) spec, void \*arr, int size)  
*Quick sort for arrays.*
- int [chooseLinearSearch](#) (const [spec\\_t](#) spec, void \*arr, int size,...)  
*Linear search for arrays.*
- void [printMatrix](#) (const [spec\\_t](#) spec, const void \*matrix, const unsigned int nRows, const unsigned int nColumns)  
*Print matrix of specified size with specified formatting.*
- void [charBubbleSort](#) (char \*arr, unsigned int size)  
*Bubblesort for arrays of chars.*
- void [intBubbleSort](#) (int \*arr, unsigned int size)  
*Bubblesort for arrays of ints.*
- void [floatBubbleSort](#) (float \*arr, unsigned int size)  
*Bubblesort for arrays of floats.*
- void [doubleBubbleSort](#) (double \*arr, unsigned int size)

- Bubblesort for arrays of doubles.*
  - void [ptrBubbleSort](#) (void \*\*arr, unsigned int size)
- Bubblesort for arrays of pointers.*
  - void [charQuickSort](#) (char \*arr, int size)
- Quicksort for arrays of chars.*
  - void [intQuickSort](#) (int \*arr, int size)
- Quicksort for arrays of ints.*
  - void [floatQuickSort](#) (float \*arr, int size)
- Quicksort for arrays of floats.*
  - void [doubleQuickSort](#) (double \*arr, int size)
- Quicksort for arrays of doubles.*
  - void [ptrQuickSort](#) (void \*\*arr, int size)
- Quicksort for arrays of pointers.*

### 5.2.1 Detailed Description

Common tasks with arrays: sorting, searching, printing etc.

Author

Pietro Firpo ( [pietro.firpo@pm.me](mailto:pietro.firpo@pm.me) )

### 5.2.2 Function Documentation

#### 5.2.2.1 charBubbleSort()

```
void charBubbleSort (  
    char * arr,  
    unsigned int size )
```

Bubblesort for arrays of chars.

Equivalent to `chooseBubbleSortArr ("%c", arr, size)`. Refer to [chooseBubbleSortArr\(\)](#)

#### 5.2.2.2 charQuickSort()

```
void charQuickSort (  
    char * arr,  
    int size )
```

Quicksort for arrays of chars.

Equivalent to `chooseQuickSortArr ("%c", arr, size)`. Refer to [chooseQuickSortArr\(\)](#)

#### 5.2.2.3 chooseBubbleSortArr()

```
void chooseBubbleSortArr (  
    const spec\_t spec,  
    void * arr,  
    unsigned int size )
```

Bubble sort for arrays.

## Parameters

<i>spec</i>	Type specifier of the array to be sorted. Refer to <a href="#">spec_t</a> for supported types.
<i>arr</i>	Pointer to the first element of the array to be sorted
<i>size</i>	Number of elements of the array to be sorted

## 5.2.2.4 chooseLinearSearch()

```
int chooseLinearSearch (
    const spec_t spec,
    void * arr,
    int size,
    ... )
```

Linear search for arrays.

## Parameters

<i>spec</i>	Type specifier of the array to be sorted. Refer to <a href="#">spec_t</a> for supported types
<i>arr</i>	Pointer to the first element of the array to be inspected
<i>size</i>	Number of elements of the array to be inspected
<i>...</i>	The key to be searched

## Note

Even though passing more than one key does not throw a compiler nor runtime error, only searching one item is supported. Other items are ignored. If you don't specify any item to be searched, still no errors occur but the return value of the function can be unpredictable

## Returns

The index of the first occurrence of the key in the array or the return code of the function

## Return values

<i>KEY_NOT_FOUND</i>	The key was not found
----------------------	-----------------------

## 5.2.2.5 chooseQuickSortArr()

```
void chooseQuickSortArr (
    const spec_t spec,
    void * arr,
    int size )
```

Quick sort for arrays.

## Parameters

<i>spec</i>	Type specifier of the array to be sorted. Refer to <a href="#">spec_t</a> for supported types
<i>arr</i>	Pointer to the first element of the array to be sorted
<i>size</i>	Number of elements of the array to be sorted

### 5.2.2.6 doubleBubbleSort()

```
void doubleBubbleSort (  
    double * arr,  
    unsigned int size )
```

Bubblesort for arrays of doubles.

Equivalent to `chooseBubbleSortArr("%lf", arr, size)`. Refer to [chooseBubbleSortArr\(\)](#)

### 5.2.2.7 doubleQuickSort()

```
void doubleQuickSort (  
    double * arr,  
    int size )
```

Quicksort for arrays of doubles.

Equivalent to `chooseQuickSortArr("%lf", arr, size)`. Refer to [chooseQuickSortArr\(\)](#)

### 5.2.2.8 floatBubbleSort()

```
void floatBubbleSort (  
    float * arr,  
    unsigned int size )
```

Bubblesort for arrays of floats.

Equivalent to `chooseBubbleSortArr("%f", arr, size)`. Refer to [chooseBubbleSortArr\(\)](#)

### 5.2.2.9 floatQuickSort()

```
void floatQuickSort (  
    float * arr,  
    int size )
```

Quicksort for arrays of floats.

Equivalent to `chooseQuickSortArr("%f", arr, size)`. Refer to [chooseQuickSortArr\(\)](#)

### 5.2.2.10 intBubbleSort()

```
void intBubbleSort (
    int * arr,
    unsigned int size )
```

Bubblesort for arrays of ints.

Equivalent to `chooseBubbleSortArr("%i", arr, size)`. Refer to [chooseBubbleSortArr\(\)](#)

### 5.2.2.11 intQuickSort()

```
void intQuickSort (
    int * arr,
    int size )
```

Quicksort for arrays of ints.

Equivalent to `chooseQuickSortArr("%i", arr, size)`. Refer to [chooseQuickSortArr\(\)](#)

### 5.2.2.12 printMatrix()

```
void printMatrix (
    const spec_t spec,
    const void * matrix,
    const unsigned int nRows,
    const unsigned int nColumns )
```

Print matrix of specified size with specified formatting.

#### Parameters

<i>spec</i>	Type and format specifier used to print a cell. The <code>printf()</code> identifier formatting convention is supported. See <a href="#">spec_t</a> for details. Additional supported specifiers: "%hi" (numerical output for char)
-------------	---

#### Note

The format specifier must end with the letter of the type specifier. For example, "%5.3lf" is supported, "%5.3lf\n" or "%5.3lfTest" is not supported and nothing is printed

#### Parameters

<i>matrix</i>	Pointer to the first element of the matrix
<i>nRows</i>	Number of rows of the matrix
<i>nColumns</i>	Number of rows of the matrix



### 5.2.2.13 ptrBubbleSort()

```
void ptrBubbleSort (
    void ** arr,
    unsigned int size )
```

Bubblesort for arrays of pointers.

Equivalent to `chooseBubbleSortArr("%p", arr, size)`. Refer to [chooseBubbleSortArr\(\)](#)

### 5.2.2.14 ptrQuickSort()

```
void ptrQuickSort (
    void ** arr,
    int size )
```

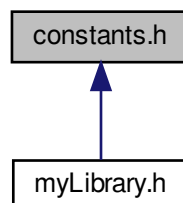
Quicksort for arrays of pointers.

Equivalent to `chooseQuickSortArr("%p", arr, size)`. Refer to [chooseQuickSortArr\(\)](#)

## 5.3 constants.h File Reference

Definition of symbolic constants used by the library.

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



## Macros

- `#define GREATER 1`  
*Returned by typeCmp() functions when first argument is greater than the second.*
- `#define EQUAL 0`  
*Returned by typeCmp() functions when first argument is equal to the second.*
- `#define SMALLER -1`  
*Returned by typeCmp() functions when first argument is smaller than the second.*
- `#define TRUE 0xFF`  
*Bool value definition.*
- `#define FALSE 0`  
*Bool value definition.*
- `#define KEY_NOT_FOUND -1`  
*Returned by search functions of the library when key was not found.*

### 5.3.1 Detailed Description

Definition of symbolic constants used by the library.

Author

Pietro Firpo ( [pietro.firpo@pm.me](mailto:pietro.firpo@pm.me) )

### 5.3.2 Macro Definition Documentation

#### 5.3.2.1 EQUAL

```
#define EQUAL 0
```

Returned by *typeCmp()* functions when first argument is equal to the second.

#### 5.3.2.2 FALSE

```
#define FALSE 0
```

Bool value definition.

#### 5.3.2.3 GREATER

```
#define GREATER 1
```

Returned by *typeCmp()* functions when first argument is greater than the second.

#### 5.3.2.4 KEY\_NOT\_FOUND

```
#define KEY_NOT_FOUND -1
```

Returned by search functions of the library when key was not found.

### 5.3.2.5 SMALLER

```
#define SMALLER -1
```

Returned by *typeCmp()* functions when first argument is smaller than the second.

### 5.3.2.6 TRUE

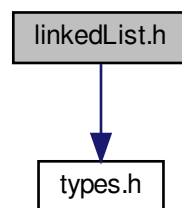
```
#define TRUE 0xFF
```

Bool value definition.

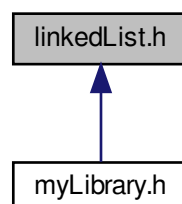
## 5.4 linkedList.h File Reference

```
#include "types.h"
```

Include dependency graph for linkedList.h:



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



## Functions

- `LinkedList newLL` (const `spec_t` spec)  
*Allocate a new `LinkedList` of specified type.*
- `LinkedList chooseNewLLFromArray` (const `spec_t` spec, const void \*arr, unsigned int size)  
*Create a `LinkedList` from an array.*
- void `printLL` (const `spec_t` spec, const `LinkedList` list)  
*Print contents from an `LinkedList`.*
- void `appendToLL` (`LinkedList` list,...)  
*Insert an item at the end of a `LinkedList`.*
- void `appendToLLFromPtr` (`LinkedList` list, void \*element)  
*Insert an item at the end of a `LinkedList`.*
- void `insertToLL` (`LinkedList` list, unsigned int index,...)  
*Insert an element at a specified position of an `LinkedList`.*
- void `deleteLL` (`LinkedList` list)  
*Delete a `LinkedList`.*
- void `getFromLL` (`LinkedList` list, unsigned int index, void \*dest)  
*Get an item from a `LinkedList`.*
- void `setLLItem` (`LinkedList` list, unsigned int index,...)  
*Set value of an element of a `LinkedList`.*
- void `removeFromLL` (`LinkedList` list, unsigned int index)  
*Remove an item from a `LinkedList`.*
- void `mergeLL` (`LinkedList` list1, const `LinkedList` list2)  
*Merge two `LinkedList`.*
- `LinkedList newLLFromLL` (const `LinkedList` list)  
*Get a copy of a `LinkedList`.*
- void `sliceLL` (`LinkedList` list, unsigned int begin, unsigned int end)  
*Slice a `LinkedList`.*
- int `linearSearchLL` (`LinkedList` list,...)  
*Linear search for `LinkedList`.*
- void \* `linearSearchLLPtr` (`LinkedList` list,...)  
*Linear search for `LinkedList`.*
- `byte areLLEqual` (const `LinkedList` list1, const `LinkedList` list2)  
*Compare two `LinkedList`.*
- `byte isInLL` (`LinkedList` list,...)  
*Detect if an element is inside a `LinkedList`.*

### 5.4.1 Function Documentation

#### 5.4.1.1 appendToLL()

```
void appendToLL (
    LinkedList list,
    ... )
```

Insert an item at the end of a `LinkedList`.

## Parameters

<i>list</i>	The <a href="#">LinkedList</a> you want to append an item to
...	The item you want to append to <code>list</code>

## Note

Even though appending more than one item does not throw a compiler nor runtime error, only appending one item is supported. Other items are ignored and are not appended to `arr`. If you don't specify any item to be appended, still no errors occur but the content of your [LinkedList](#) can be messed up

5.4.1.2 `appendToLLFromPtr()`

```
void appendToLLFromPtr (
    LinkedList list,
    void * element )
```

Insert an item at the end of a [LinkedList](#).

## Parameters

<i>list</i>	The <a href="#">LinkedList</a> you want to append an item to
<i>element</i>	Pointer to the item you want to append to <code>list</code>

5.4.1.3 `areLLEqual()`

```
byte areLLEqual (
    const LinkedList list1,
    const LinkedList list2 )
```

Compare two [LinkedList](#).

## Parameters

<i>list1</i>	The first <a href="#">LinkedList</a> you want to compare
<i>list2</i>	The second <a href="#">LinkedList</a> you want to compare

## Returns

The result of the comparison

## Return values

<i>TRUE</i>	<code>list1</code> and <code>list2</code> have equal type, equal length and equal contents
<i>FALSE</i>	<code>list1</code> and <code>list2</code> do not have equal type, equal length or equal contents

#### 5.4.1.4 chooseNewLLFromArray()

```
LinkedList chooseNewLLFromArray (
    const spec_t spec,
    const void * arr,
    unsigned int size )
```

Create a [LinkedList](#) from an array.

##### Parameters

<i>spec</i>	The type specifier of the array passed. Refer to <a href="#">spec_t</a>
<i>arr</i>	The array you want to create the <a href="#">LinkedList</a> from
<i>size</i>	The number of items of <i>arr</i>

##### Returns

A [LinkedList](#) containing the elements in *arr* in the same order

#### 5.4.1.5 deleteLL()

```
void deleteLL (
    LinkedList list )
```

Delete a [LinkedList](#).

##### Parameters

<i>list</i>	The <a href="#">LinkedList</a> you want to delete
-------------	---

#### 5.4.1.6 getFromLL()

```
void getFromLL (
    LinkedList list,
    unsigned int index,
    void * dest )
```

Get an item from a [LinkedList](#).

##### Parameters

<i>arr</i>	The <a href="#">LinkedList</a> you want to get an item from
<i>index</i>	The index of the item you want to get
<i>dest</i>	The address of the variable you want to store the item in

#### 5.4.1.7 insertToLL()

```
void insertToLL (  
    LinkedList list,  
    unsigned int index,  
    ... )
```

Insert an element at a specified position of an [LinkedList](#).

##### Parameters

<i>list</i>	The <a href="#">LinkedList</a> you want to insert an element into
<i>index</i>	The position you want to insert element at
...	The item you want to insert into list

##### Note

Even though inserting more than one item does not throw a compiler nor runtime error, only inserting one item is supported. Other items are ignored and are not inserted into `arr`. If you don't specify any item to be inserted, still no errors occur but the content of your [LinkedList](#) can be messed up

#### 5.4.1.8 isInLL()

```
byte isInLL (  
    LinkedList list,  
    ... )
```

Detect if an element is inside a [LinkedList](#).

##### Parameters

<i>list</i>	The <a href="#">LinkedList</a> you want search in
...	The element you want to search

##### Note

Even though inserting zero more than one item does not throw a compiler nor runtime error, only searching one item is supported. Other items are ignored. If you don't specify any item to be searched, still no errors occur but the return value of the function can be unpredictable

##### Return values

<i>TRUE</i>	Given element is contained in list
<i>FALSE</i>	Given element is not contained in list

#### 5.4.1.9 linearSearchLL()

```
int linearSearchLL (
    LinkedList list,
    ... )
```

Linear search for [LinkedList](#).

##### Parameters

<i>list</i>	The <a href="#">LinkedList</a> to be inspected
...	The key to be searched

##### Note

This function does not support float and double [LinkedList](#)

Even though passing more than one key does not throw a compiler nor runtime error, only searching one item is supported. Other items are ignored. If you don't specify any item to be searched, still no errors occur but the return value of the function can be unpredictable

##### Returns

The index of the first occurrence of the key in the list or the return code of the function

##### Return values

<i>KEY_NOT_FOUND</i>	The key was not found
----------------------	-----------------------

#### 5.4.1.10 linearSearchLLPtr()

```
void* linearSearchLLPtr (
    LinkedList list,
    ... )
```

Linear search for [LinkedList](#).

##### Parameters

<i>list</i>	The <a href="#">LinkedList</a> to be inspected
...	The key to be searched



**Note**

This function does not support float and double [LinkedList](#)

Even though passing more than one key does not throw a compiler nor runtime error, only searching one item is supported. Other items are ignored. If you don't specify any item to be searched, still no errors occur but the return value of the function can be unpredictable

**Returns**

A void pointer of the first occurrence of the key in the list or the return code of the function

**Return values**

<code>NULL</code>	The key was not found
-------------------	-----------------------

**5.4.1.11 `mergeLL()`**

```
void mergeLL (
    LinkedList list1,
    const LinkedList list2 )
```

Merge two [LinkedList](#).

**Parameters**

<i>list1</i>	The first <a href="#">LinkedList</a> to be merged, where the merged <a href="#">LinkedList</a> is saved
<i>list2</i>	The second <a href="#">LinkedList</a> to be merged

**5.4.1.12 `newLL()`**

```
LinkedList newLL (
    const spec\_t spec )
```

Allocate a new [LinkedList](#) of specified type.

**Parameters**

<i>spec</i>	Type specifier of the <a href="#">LinkedList</a> you want to create
-------------	---

**Returns**

An empty [LinkedList](#)

#### 5.4.1.13 newLLFromLL()

```
LinkedList newLLFromLL (
    const LinkedList list )
```

Get a copy of a [LinkedList](#).

##### Parameters

<i>list</i>	The <a href="#">LinkedList</a> you want to copy
-------------	---

##### Returns

A copy of `list`

#### 5.4.1.14 printLL()

```
void printLL (
    const spec_t spec,
    const LinkedList list )
```

Print contents from an [LinkedList](#).

##### Parameters

<i>spec</i>	The type and format specifier you want to use to print the single element of the <a href="#">LinkedList</a>
<i>list</i>	The <a href="#">LinkedList</a> you want to print

#### 5.4.1.15 removeFromLL()

```
void removeFromLL (
    LinkedList list,
    unsigned int index )
```

Remove an item from a [LinkedList](#).

##### Parameters

<i>list</i>	The <a href="#">LinkedList</a> you want to delete an item from
<i>index</i>	The index of the item you want to delete

#### 5.4.1.16 setLLItem()

```
void setLLItem (
    LinkedList list,
    unsigned int index,
    ... )
```

Set value of an element of a [LinkedList](#).

##### Parameters

<i>list</i>	The <a href="#">LinkedList</a> you want to edit
<i>index</i>	The index of the element you want to change
<i>...</i>	The item you want to insert into <i>list</i>

##### Note

Even though inserting more than one item does not throw a compiler nor runtime error, only setting one item is supported. Other items are ignored. If you don't specify any item to be inserted, still no errors occur but the content of your [LinkedList](#) can be messed up

#### 5.4.1.17 sliceLL()

```
void sliceLL (
    LinkedList list,
    unsigned int begin,
    unsigned int end )
```

Slice a [LinkedList](#).

##### Parameters

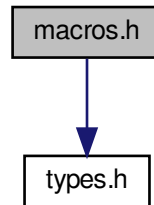
<i>list</i>	The <a href="#">LinkedList</a> you want to slice, where the sliced <a href="#">LinkedList</a> is saved
<i>begin</i>	The index of the beginning of the slice
<i>end</i>	The index of the end of the slice

## 5.5 macros.h File Reference

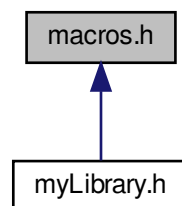
Macros for emulated overloading.

```
#include "types.h"
```

Include dependency graph for macros.h:



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



## Macros

- `#define cmpVal(a, b) _Generic((a, b), char: charCmp, int: intCmp, float: floatCmp, double: doubleCmp, void *: ptrCmp)(&a, &b)`  
*Compare two values. Calls the right typeCmp() function.*
- `#define bubbleSortArr(arr, size) _Generic(arr, char *: charBubbleSort, int *: intBubbleSort, float *: floatBubbleSort, double *: doubleBubbleSort, void **: ptrBubbleSort)(arr, size)`  
*BubbleSort for arrays.*
- `#define quickSortArr(arr, size) _Generic(arr, char *: charQuickSort, int *: intQuickSort, float *: floatQuickSort, double *: doubleQuickSort, void **: ptrQuickSort)(arr, size)`  
*Quicksort for arrays.*

### 5.5.1 Detailed Description

Macros for emulated overloading.

**Author**

Pietro Firpo ( [pietro.firpo@pm.me](mailto:pietro.firpo@pm.me) )

**Note**

Many of these macros work on C11 or newer compilers only. If they are not supported by your compiler you have to use the function the macro expands to in your case. For example, if you want to bubblesort an array of floats and the macro `bubbleSort()` is not supported by your compiler, you have to call `floatBubbleSort()` or `chooseBubbleSortArr()`

In some development environments, for example Vscode, calls to these macros can be reported as errors even if they are correct. If you use Vscode you have to set `"C_Cpp.default.cStandard": "c17"` in your `settings.json` file in order to avoid this error reportings

## 5.5.2 Macro Definition Documentation

### 5.5.2.1 bubbleSortArr

```
#define bubbleSortArr(  
    arr,  
    size ) _Generic(arr, char *: charBubbleSort, int *: intBubbleSort, float *↔  
: floatBubbleSort, double *: doubleBubbleSort, void **: ptrBubbleSort)(arr, size)
```

BubbleSort for arrays.

**Returns**

The return code of the function called

**Parameters**

<i>arr</i>	Pointer to the array to be sorted
<i>size</i>	Number of elements in the array to be sorted

### 5.5.2.2 cmpVal

```
#define cmpVal(  
    a,  
    b ) _Generic((a, b), char: charCmp, int: intCmp, float: floatCmp, double↔  
: doubleCmp, void *: ptrCmp)(&a, &b)
```

Compare two values. Calls the right `typeCmp()` function.

**Note**

This macro must be called on variables. For example, `cmpVal(2, 3)` is not supported

**Parameters**

<i>a</i>	First value to be compared
<i>b</i>	Second value to be compared

**Returns**

The return code of the function called

**Return values**

<i>GREATER</i>	First element is grater than the second
<i>EQUAL</i>	First element is equal to the second
<i>SMALLER</i>	First element is smaller than the second

**5.5.2.3 quickSortArr**

```
#define quickSortArr(
    arr,
    size ) _Generic(arr, char *: charQuickSort, int *: intQuickSort, float *↔
:floatQuickSort, double *: doubleQuickSort, void **: ptrQuickSort)(arr, size)
```

Quicksort for arrays.

**Returns**

The return code of the function called

**Parameters**

<i>arr</i>	Pointer to the array to be sorted
<i>size</i>	Number of elements in the array to be sorted

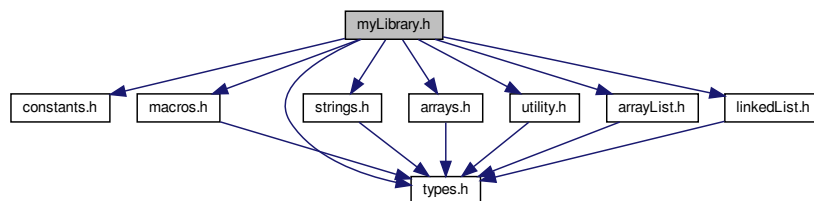
**5.6 myLibrary.h File Reference**

Includes all other headers. Useful for rapid import.

```
#include "constants.h"
#include "macros.h"
#include "types.h"
#include "strings.h"
#include "arrays.h"
#include "utility.h"
#include "arrayList.h"
```

```
#include "linkedList.h"
```

Include dependency graph for myLibrary.h:



### 5.6.1 Detailed Description

Includes all other headers. Useful for rapid import.

Author

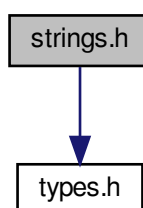
Pietro Firpo ( [pietro.firpo@pm.me](mailto:pietro.firpo@pm.me) )

## 5.7 strings.h File Reference

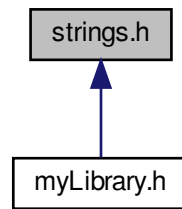
Common tasks with strings.

```
#include "types.h"
```

Include dependency graph for strings.h:



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



## Functions

- `string getString ()`  
*Reads from terminal a string of arbitrary length.*
- `byte endsWith (const string str, const string suffix)`  
*Check if a string ends with the specified substring.*
- `string changeLastCharacter (const string str, char newCharacter)`  
*Get string with different last character.*
- `unsigned int getLength (const string str)`  
*Get the lenght of a string.*
- `string copyOf (const string src)`  
*Get a copy of the given string.*

### 5.7.1 Detailed Description

Common tasks with strings.

Author

Pietro Firpo ( [pietro.firpo@pm.me](mailto:pietro.firpo@pm.me) )

### 5.7.2 Function Documentation

#### 5.7.2.1 changeLastCharacter()

```
string changeLastCharacter (  
    const string str,  
    char newCharacter )
```

Get string with different last character.



## Parameters

<i>str</i>	The string you want to change the last character
<i>newCharacter</i>	The character you want to set as last character

## Returns

A pointer to a string with the same characters of `str` and `newCharacter` as last character or a null pointer in case of errors

### 5.7.2.2 copyOf()

```
string copyOf (
    const string src )
```

Get a copy of the given string.

## Parameters

<i>src</i>	The string to be copied
------------	-------------------------

## Returns

A pointer to the copy of the given string or or a null pointer in case of errors

### 5.7.2.3 endsWith()

```
byte endsWith (
    const string str,
    const string suffix )
```

Check if a string ends with the specified substring.

## Parameters

<i>str</i>	The string to be inspected
<i>suffix</i>	The string you want to check if <code>string</code> ends with

## Returns

A boolean value

## Return values

<i>TRUE</i>	<code>str</code> ends with <code>suffix</code>
<i>FALSE</i>	<code>str</code> does not end with <code>suffix</code>

5.7.2.4 `getLength()`

```
unsigned int getLength (
    const string str )
```

Get the lenght of a string.

## Parameters

<i>str</i>	The string to be evaluated
------------	----------------------------

## Returns

The lenght of the given string (terminator EXCLUDED) or the return code of the function

5.7.2.5 `getString()`

```
string getString ( )
```

Reads from terminal a string of arbitrary length.

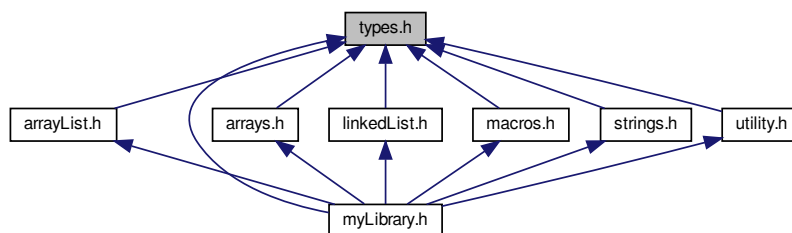
## Returns

A char pointer to the first element of the string or a null pointer in case of errors

5.8 `types.h` File Reference

Collection of useful types.

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



## Data Structures

- struct [ArrayList](#)  
*ArrayList type*
- struct [node](#)  
*Node type*
- struct [LinkedList](#)  
*LinkedList type*

## Typedefs

- typedef char [byte](#)  
*Alias for char, just to avoid confusion with 8 bit numbers and ASCII characters.*
- typedef char \* [spec\\_t](#)  
*Used to specify type of argument passed in functions that require a type specifier.*
- typedef char \* [string](#)  
*Alias for char \*, used when an array of char is actually used as a string.*
- typedef struct [node](#) \* [Node](#)  
*Node type*

### 5.8.1 Detailed Description

Collection of useful types.

Author

Pietro Firpo ( [pietro.firpo@pm.me](mailto:pietro.firpo@pm.me) )

### 5.8.2 Typedef Documentation

#### 5.8.2.1 byte

```
typedef char byte
```

Alias for char, just to avoid confusion with 8 bit numbers and ASCII characters.

#### 5.8.2.2 Node

```
typedef struct node * Node
```

[Node](#) type

Base component of every linked data type

Note

All the parameters in this structure must be intended as read-only. Manually modifying them can cause unknown and unwanted behavior

### 5.8.2.3 spec\_t

```
typedef char* spec_t
```

Used to specify type of argument passed in functions that require a type specifier.

Supported specifiers: "%c" (char), "%i" (int), "%f" (float), "%lf" (double), "%p" (pointer)

#### Note

Some functions may not support some identifiers or may support additional identifiers. In those cases refer to that function documentation

### 5.8.2.4 string

```
typedef char* string
```

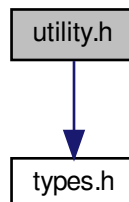
Alias for char \*, used when an array of char is actually used as a string.

## 5.9 utility.h File Reference

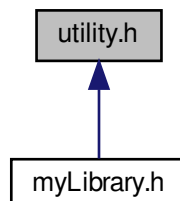
Common tasks such as comparing variables, swap bools, allocate memory.

```
#include "types.h"
```

Include dependency graph for utility.h:



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



## Functions

- [byte chooseCmp](#) (const [spec\\_t](#) spec, const void \*a, const void \*b)  
*Compare two chars.*
- [byte charCmp](#) (const void \*a, const void \*b)  
*Compare two chars.*
- [byte byteCmp](#) (const void \*a, const void \*b)  
*Compare two bytes.*
- [byte intCmp](#) (const void \*a, const void \*b)  
*Compare two ints.*
- [byte floatCmp](#) (const void \*a, const void \*b)  
*Compare two floats.*
- [byte doubleCmp](#) (const void \*a, const void \*b)  
*Compare two doubles.*
- [byte ptrCmp](#) (const void \*a, const void \*b)  
*Compare two pointers.*
- void \* [saferMalloc](#) (unsigned int bytes)  
*Return a pointer to a space in memory of specified size.*
- void \* [saferRealloc](#) (void \*pointer, unsigned int bytes)  
*Reallocate a space in memory.*

### 5.9.1 Detailed Description

Common tasks such as comparing variables, swap bools, allocate memory.

Author

Pietro Firpo ( [pietro.firpo@pm.me](mailto:pietro.firpo@pm.me) )

### 5.9.2 Function Documentation

#### 5.9.2.1 byteCmp()

```
byte byteCmp (  
    const void * a,  
    const void * b )
```

Compare two bytes.

Equivalent to `charCmp(a, b)`. Refer to [charCmp\(\)](#).

#### 5.9.2.2 charCmp()

```
byte charCmp (  
    const void * a,  
    const void * b )
```

Compare two chars.

Equivalent to `chooseCmp("%c", a, b)`. Refer to [chooseCmp\(\)](#)

### 5.9.2.3 chooseCmp()

```
byte chooseCmp (
    const spec_t spec,
    const void * a,
    const void * b )
```

Compare two chars.

#### Parameters

<i>spec</i>	Type specifier of the values to be sorted. Refer to <a href="#">spec_t</a> for supported types.
<i>a</i>	Pointer to the first element to be compared
<i>b</i>	Pointer to the second element to be compared

#### Returns

Constant for the corresponding comparison result

#### Return values

<i>GREATER</i>	First element is greater than the second
<i>EQUAL</i>	First element is equal to the second
<i>SMALLER</i>	First element is smaller than the second

### 5.9.2.4 doubleCmp()

```
byte doubleCmp (
    const void * a,
    const void * b )
```

Compare two doubles.

Equivalent to `chooseCmp ("%lf", a, b)`. Refer to [chooseCmp\(\)](#)

### 5.9.2.5 floatCmp()

```
byte floatCmp (
    const void * a,
    const void * b )
```

Compare two floats.

Equivalent to `chooseCmp ("%f", a, b)`. Refer to [chooseCmp\(\)](#)

### 5.9.2.6 intCmp()

```
byte intCmp (
    const void * a,
    const void * b )
```

Compare two ints.

Equivalent to `chooseCmp ("%i", a, b)`. Refer to [chooseCmp\(\)](#)

### 5.9.2.7 ptrCmp()

```
byte ptrCmp (
    const void * a,
    const void * b )
```

Compare two pointers.

Equivalent to `chooseCmp ("%p", a, b)`. Refer to [chooseCmp\(\)](#)

### 5.9.2.8 saferMalloc()

```
void* saferMalloc (
    unsigned int bytes )
```

Return a pointer to a space in memory of specified size.

Calls `malloc(bytes)` for a maximum of 10 times until it returns a not null pointer. If in 10 calls does not manage to obtain a not null pointer makes the program terminate

#### Parameters

<i>bytes</i>	Number of bytes to allocate
--------------	-----------------------------

#### Returns

A pointer to the allocated memory

### 5.9.2.9 saferRealloc()

```
void* saferRealloc (
    void * pointer,
    unsigned int bytes )
```

Reallocate a space in memory.

Calls `realloc(pointer, bytes)` for a maximum of 10 times until it returns a not null pointer. If in 10 calls does not manage to obtain a not null pointer makes the program terminate

**Parameters**

<i>pointer</i>	Pointer to the memory to be reallocated
<i>bytes</i>	Number of bytes to allocate

**Returns**

A pointer to the allocated memory



# Index

- appendToAL
  - arrayList.h, [13](#)
- appendToLL
  - linkedList.h, [30](#)
- appendToLLFromPtr
  - linkedList.h, [31](#)
- areALEqual
  - arrayList.h, [14](#)
- areLLEqual
  - linkedList.h, [31](#)
- ArrayList, [7](#)
  - body, [7](#)
  - size, [7](#)
  - type, [8](#)
- arrayList.h, [11](#)
  - appendToAL, [13](#)
  - areALEqual, [14](#)
  - bubbleSortAL, [14](#)
  - chooseNewALFromArray, [15](#)
  - deleteAL, [15](#)
  - getFromAL, [15](#)
  - insertToAL, [16](#)
  - isInAL, [16](#)
  - linearSearchAL, [17](#)
  - mergeAL, [17](#)
  - newAL, [17](#)
  - newALFromAL, [18](#)
  - newALFromArray, [13](#)
  - newALFromByteArray, [18](#)
  - newALFromCharArray, [18](#)
  - newALFromDoubleArray, [18](#)
  - newALFromFloatArray, [19](#)
  - newALFromIntArray, [19](#)
  - newALFromPtrArray, [19](#)
  - printAL, [19](#)
  - quickSortAL, [20](#)
  - removeFromAL, [20](#)
  - reverseAL, [20](#)
  - setALItem, [20](#)
  - sliceAL, [21](#)
- arrays.h, [21](#)
  - charBubbleSort, [23](#)
  - charQuickSort, [23](#)
  - chooseBubbleSortArr, [23](#)
  - chooseLinearSearch, [24](#)
  - chooseQuickSortArr, [24](#)
  - doubleBubbleSort, [25](#)
  - doubleQuickSort, [25](#)
  - floatBubbleSort, [25](#)
  - floatQuickSort, [25](#)
  - intBubbleSort, [25](#)
  - intQuickSort, [26](#)
  - printMatrix, [26](#)
  - ptrBubbleSort, [26](#)
  - ptrQuickSort, [27](#)
- body
  - ArrayList, [7](#)
- bubbleSortAL
  - arrayList.h, [14](#)
- bubbleSortArr
  - macros.h, [39](#)
- byte
  - types.h, [45](#)
- byteCmp
  - utility.h, [47](#)
- changeLastCharacter
  - strings.h, [42](#)
- charBubbleSort
  - arrays.h, [23](#)
- charCmp
  - utility.h, [47](#)
- charQuickSort
  - arrays.h, [23](#)
- chooseBubbleSortArr
  - arrays.h, [23](#)
- chooseCmp
  - utility.h, [47](#)
- chooseLinearSearch
  - arrays.h, [24](#)
- chooseNewALFromArray
  - arrayList.h, [15](#)
- chooseNewLLFromArray
  - linkedList.h, [32](#)
- chooseQuickSortArr
  - arrays.h, [24](#)
- cmpVal
  - macros.h, [39](#)
- constants.h, [27](#)
  - EQUAL, [28](#)
  - FALSE, [28](#)
  - GREATER, [28](#)
  - KEY\_NOT\_FOUND, [28](#)
  - SMALLER, [28](#)
  - TRUE, [29](#)
- copyOf
  - strings.h, [43](#)

- data
  - node, [10](#)
- deleteAL
  - arrayList.h, [15](#)
- deleteLL
  - linkedList.h, [32](#)
- doubleBubbleSort
  - arrays.h, [25](#)
- doubleCmp
  - utility.h, [48](#)
- doubleQuickSort
  - arrays.h, [25](#)
- endsWith
  - strings.h, [43](#)
- EQUAL
  - constants.h, [28](#)
- FALSE
  - constants.h, [28](#)
- floatBubbleSort
  - arrays.h, [25](#)
- floatCmp
  - utility.h, [48](#)
- floatQuickSort
  - arrays.h, [25](#)
- getFromAL
  - arrayList.h, [15](#)
- getFromLL
  - linkedList.h, [32](#)
- getLength
  - strings.h, [44](#)
- getString
  - strings.h, [44](#)
- GREATER
  - constants.h, [28](#)
- head
  - LinkedList, [9](#)
- insertToAL
  - arrayList.h, [16](#)
- insertToLL
  - linkedList.h, [33](#)
- intBubbleSort
  - arrays.h, [25](#)
- intCmp
  - utility.h, [48](#)
- intQuickSort
  - arrays.h, [26](#)
- isInAL
  - arrayList.h, [16](#)
- isInLL
  - linkedList.h, [33](#)
- KEY\_NOT\_FOUND
  - constants.h, [28](#)
- linearSearchAL
  - arrayList.h, [17](#)
- linearSearchLL
  - linkedList.h, [34](#)
- linearSearchLLPtr
  - linkedList.h, [34](#)
- linked
  - node, [10](#)
- LinkedList, [8](#)
  - head, [9](#)
  - size, [9](#)
  - tail, [9](#)
  - type, [9](#)
- linkedList.h, [29](#)
  - appendToLL, [30](#)
  - appendToLLFromPtr, [31](#)
  - areLLEqual, [31](#)
  - chooseNewLLFromArray, [32](#)
  - deleteLL, [32](#)
  - getFromLL, [32](#)
  - insertToLL, [33](#)
  - isInLL, [33](#)
  - linearSearchLL, [34](#)
  - linearSearchLLPtr, [34](#)
  - mergeLL, [35](#)
  - newLL, [35](#)
  - newLLFromLL, [35](#)
  - printLL, [36](#)
  - removeFromLL, [36](#)
  - setLLItem, [36](#)
  - sliceLL, [37](#)
- macros.h, [37](#)
  - bubbleSortArr, [39](#)
  - cmpVal, [39](#)
  - quickSortArr, [40](#)
- mergeAL
  - arrayList.h, [17](#)
- mergeLL
  - linkedList.h, [35](#)
- myLibrary.h, [40](#)
- newAL
  - arrayList.h, [17](#)
- newALFromAL
  - arrayList.h, [18](#)
- newALFromArray
  - arrayList.h, [13](#)
- newALFromByteArray
  - arrayList.h, [18](#)
- newALFromCharArray
  - arrayList.h, [18](#)
- newALFromDoubleArray
  - arrayList.h, [18](#)
- newALFromFloatArray
  - arrayList.h, [19](#)
- newALFromIntArray
  - arrayList.h, [19](#)
- newALFromPtrArray
  - arrayList.h, [19](#)

- newLL
  - linkedList.h, [35](#)
- newLLFromLL
  - linkedList.h, [35](#)
- Node
  - types.h, [45](#)
- node, [10](#)
  - data, [10](#)
  - linked, [10](#)
- printAL
  - arrayList.h, [19](#)
- printLL
  - linkedList.h, [36](#)
- printMatrix
  - arrays.h, [26](#)
- ptrBubbleSort
  - arrays.h, [26](#)
- ptrCmp
  - utility.h, [49](#)
- ptrQuickSort
  - arrays.h, [27](#)
- quickSortAL
  - arrayList.h, [20](#)
- quickSortArr
  - macros.h, [40](#)
- removeFromAL
  - arrayList.h, [20](#)
- removeFromLL
  - linkedList.h, [36](#)
- reverseAL
  - arrayList.h, [20](#)
- saferMalloc
  - utility.h, [49](#)
- saferRealloc
  - utility.h, [49](#)
- setALItem
  - arrayList.h, [20](#)
- setLLItem
  - linkedList.h, [36](#)
- size
  - ArrayList, [7](#)
  - LinkedList, [9](#)
- sliceAL
  - arrayList.h, [21](#)
- sliceLL
  - linkedList.h, [37](#)
- SMALLER
  - constants.h, [28](#)
- spec\_t
  - types.h, [45](#)
- string
  - types.h, [46](#)
- strings.h, [41](#)
  - changeLastCharacter, [42](#)
  - copyOf, [43](#)
  - endsWith, [43](#)
  - getLength, [44](#)
  - getString, [44](#)
- tail
  - LinkedList, [9](#)
- TRUE
  - constants.h, [29](#)
- type
  - ArrayList, [8](#)
  - LinkedList, [9](#)
- types.h, [44](#)
  - byte, [45](#)
  - Node, [45](#)
  - spec\_t, [45](#)
  - string, [46](#)
- utility.h, [46](#)
  - byteCmp, [47](#)
  - charCmp, [47](#)
  - chooseCmp, [47](#)
  - doubleCmp, [48](#)
  - floatCmp, [48](#)
  - intCmp, [48](#)
  - ptrCmp, [49](#)
  - saferMalloc, [49](#)
  - saferRealloc, [49](#)