# My library

Generated by Doxygen 1.9.1

# 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:

# Chapter 3

# File Index

## 3.1 File List

Here is a list of all files with brief descriptions:

# 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:

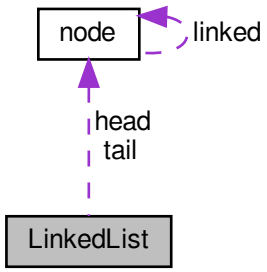
- types.h

## 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:

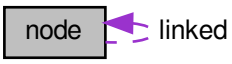
- types.h

## 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

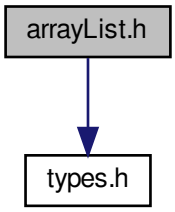
# File Documentation

## 5.1 arrayList.h File Reference

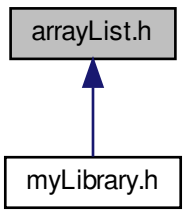
ArrayList functions and macros

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

Include dependency graph for arrayList.h:




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



## Macros

- #define newALFromArray(arr, size)

  *Create an ArrayList from a static array.*

## Functions

- ArrayList newAL (const spec_t spec)

  *Allocate a new ArrayList of specified type.*
- ArrayList newALFromAL (const ArrayList arr)

  *Get a copy of an ArrayList.*
- void appendToAL (ArrayList arr,...)

  *Insert an item at the end of an ArrayList.*
- void insertToAL (ArrayList arr, unsigned int index,...)

  *Insert an element at a specified position of an ArrayList.*
- void setALItem (ArrayList arr, unsigned int index,...)

  *Set value of an element of an ArrayList.*
- void mergeAL (ArrayList arr1, const ArrayList arr2)

  *Merge two ArrayList.*
- void sliceAL (ArrayList arr, unsigned int begin, unsigned int end)

  *Slice an ArrayList.*
- void printAL (const spec_t spec, const ArrayList arr)

  *Print contents from an ArrayList.*
- void removeFromAL (ArrayList arr, unsigned int index)

  *Remove an item from an ArrayList.*
- void getFromAL (const ArrayList arr, unsigned int index, void *dest)

  *Get an item from an ArrayList.*
- void deleteAL (ArrayList arr)

  *Delete an ArrayList.*
- byte areALEqual (const ArrayList arr1, const ArrayList arr2)

  *Compare two ArrayList.*
- void reverseAL (ArrayList arr)

  *Reverse an ArrayList.*
- void bubbleSortAL (ArrayList arr)

  *Bubble sort for ArrayList.*
- void quickSortAL (ArrayList arr)

  *Quicksort for ArrayList.*
- byte isInAL (ArrayList arr,...)

  *Detect if an element is inside an ArrayList.*
- int linearSearchAL (ArrayList arr,...)

  *Linear search for ArrayList.*
- ArrayList chooseNewALFromArray (const spec_t spec, const void *arr, unsigned int size)

  *Create an ArrayList from an array.*
- ArrayList newALFromCharArray (const char arr[ ], unsigned int size)

  *Create ArrayList from an array of chars.*
- ArrayList newALFromByteArray (const char arr[ ], unsigned int size)

  *Alias for newALFromCharArray(). Used to create ArrayList from byte array. Refer to newALFromCharArray()*
- ArrayList newALFromIntArray (const int arr[ ], unsigned int size)

  *Create ArrayList from an array of ints.*
- ArrayList newALFromFloatArray (const float arr[ ], unsigned int size)

*Create ArrayList from an array of floats.*

- ArrayList newALFromDoubleArray (const double arr[ ], unsigned int size)

  *Create ArrayList from an array of doubles.*

- ArrayList newALFromPtrArray (const void *arr, unsigned int size)

  *Create ArrayList from an array of pointers.*

## 5.1.1 Detailed Description

ArrayList functions and macros

**Author**

Pietro Firpo ( pietro.firpo@pm.me)

## 5.1.2 Macro Definition Documentation

### 5.1.2.1 newALFromArray

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

**Value:**

```
_Generic(arr, char *                        \
: newALFromCharArray, int *      \
: newALFromIntArray, float *     \
: newALFromFloatArray, double * \
: newALFromDoubleArray)(arr, size)
```

Create an ArrayList from a static array.

**Parameters**

| | |
|---|---|
| *arr* | The array you want to create an ArrayList from |
| *size* | The size of `arr` |

**Note**

Passing an array of pointers is not supported

**Returns**

An ArrayList containing all the elements of `arr`

## 5.1.3 Function Documentation

#### 5.1.3.1 appendToAL()

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

Insert an item at the end of an ArrayList.

**Parameters**

| | |
|---|---|
| *arr* | The ArrayList you want to append an item to |
| *...* | The item you want to append to `arr` |

**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 ArrayList can be messed up

#### 5.1.3.2 areALEqual()

```
byte areALEqual (
            const ArrayList arr1,
            const ArrayList arr2 )
```

Compare two ArrayList.

**Parameters**

| | |
|---|---|
| *arr1* | The first ArrayList you want to compare |
| *arr2* | The second ArrayList you want to compare |

**Returns**

The result of the comparison

**Return values**

| | |
|---|---|
| *TRUE* | `arr1` and `arr2` have equal type, equal length and equal contents |
| *FALSE* | `arr1` and `arr2` do not have equal type, equal length or equal contents |

#### 5.1.3.3 bubbleSortAL()

```
void bubbleSortAL (
            ArrayList arr )
```

Bubble sort for ArrayList.

**Parameters**

| | |
|---|---|
| *arr* | The ArrayList you want to bubble sort |

#### 5.1.3.4 chooseNewALFromArray()

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

Create an ArrayList from an array.

**Parameters**

| | |
|---|---|
| *spec* | The type specifier of the array passed. Refer to spec_t |
| *arr* | The array you want to create the ArrayList from |
| *size* | The number of items of `arr` |

**Returns**

An ArrayList containing the elements in `arr` in the same order

#### 5.1.3.5 deleteAL()

```
void deleteAL (
            ArrayList arr )
```

Delete an ArrayList.

**Parameters**

| | |
|---|---|
| *arr* | The ArrayList you want to delete |

#### 5.1.3.6 getFromAL()

```
void getFromAL (
            const ArrayList arr,
```

```
            unsigned int index,
            void * dest )
```

Get an item from an ArrayList.

**Parameters**

| | |
|---|---|
| *arr* | The ArrayList you want to get an item from |
| *index* | The index of the item you want to get |
| *dest* | The address of the variable you want to store the item in |

#### 5.1.3.7 insertToAL()

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

Insert an element at a specified position of an ArrayList.

**Parameters**

| | |
|---|---|
| *arr* | The ArrayList you want to insert an element into |
| *index* | The position you want to insert `element` at |
| *...* | The item you want to insert into `arr` |

**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 ArrayList can be messed up

#### 5.1.3.8 isInAL()

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

Detect if an element is inside an ArrayList.

**Parameters**

| | |
|---|---|
| *arr* | The ArrayList 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**

| | |
|---|---|
| *TRUE* | Given element is contained in `arr` |
| *FALSE* | Given element is not contained in `arr` |

#### 5.1.3.9 linearSearchAL()

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

Linear search for ArrayList.

**Parameters**

| | |
|---|---|
| *arr* | The ArrayList to be inspected |
| *...* | 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 occurence of the key in the array or the return code of the function

**Return values**

| | |
|---|---|
| *KEY_NOT_FOUND* | The key was not found |

#### 5.1.3.10 mergeAL()

```
void mergeAL (
            ArrayList arr1,
            const ArrayList arr2 )
```

Merge two ArrayList.

**Parameters**

| | |
|---|---|
| *arr1* | The first ArrayList to be merged, where the merged ArrayList is saved |
| *arr2* | The second ArrayList to be merged |

#### 5.1.3.11 newAL()

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

Allocate a new ArrayList of specified type.

**Parameters**

| | |
|---|---|
| *spec* | Type specifier of the ArrayList you want to create |

**Returns**

An empty ArrayList

#### 5.1.3.12 newALFromAL()

```
ArrayList newALFromAL (
            const ArrayList arr )
```

Get a copy of an ArrayList.

**Parameters**

| | |
|---|---|
| *arr* | The ArrayList you want to copy |

**Returns**

A copy of `arr`

#### 5.1.3.13 newALFromByteArray()

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

Alias for newALFromCharArray(). Used to create ArrayList from byte array. Refer to newALFromCharArray()

#### 5.1.3.14 newALFromCharArray()

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

Create ArrayList from an array of chars.

Equivalent to `chooseNewALFromArray("%c", arr, size)`. Refer to chooseNewALFromArray()

#### 5.1.3.15 newALFromDoubleArray()

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

Create ArrayList from an array of doubles.

Equivalent to `chooseNewALFromArray("%lf", arr, size)`. Refer to chooseNewALFromArray()

#### 5.1.3.16 newALFromFloatArray()

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

Create ArrayList from an array of floats.

Equivalent to `chooseNewALFromArray("%f", arr, size)`. Refer to chooseNewALFromArray()

#### 5.1.3.17 newALFromIntArray()

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

Create ArrayList from an array of ints.

Equivalent to `chooseNewALFromArray("%i", arr, size)`. Refer to chooseNewALFromArray()

#### 5.1.3.18 newALFromPtrArray()

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

Create ArrayList from an array of pointers.

Equivalent to `chooseNewALFromArray("%p", arr, size)`. Refer to chooseNewALFromArray()

#### 5.1.3.19 printAL()

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

Print contents from an ArrayList.

**Parameters**

| | |
|---|---|
| *spec* | The type and format specifier you want to use to print the single element of the ArrayList |
| *arr* | The ArrayList you want to print |

#### 5.1.3.20 quickSortAL()

```
void quickSortAL (
            ArrayList arr )
```

Quicksort for ArrayList.

**Parameters**

| | |
|---|---|
| *arr* | The ArrayList you want to quicksort |

#### 5.1.3.21 removeFromAL()

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

Remove an item from an ArrayList.

**Parameters**

| | |
|---|---|
| *arr* | The ArrayList you want to delete an item from |
| *index* | The index of the item you want to delete |

#### 5.1.3.22 reverseAL()

```
void reverseAL (
            ArrayList arr )
```

Reverse an ArrayList.

**Parameters**

| | |
|---|---|
| *arr* | The ArrayList you want to reverse |

#### 5.1.3.23 setALItem()

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

Set value of an element of an ArrayList.

**Parameters**

| | |
|---|---|
| *arr* | The ArrayList you want to edit |
| *index* | The index of the element you want to change |
| *...* | The item you want to insert into `arr` |

**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

#### 5.1.3.24 sliceAL()

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

Slice an ArrayList.

**Parameters**

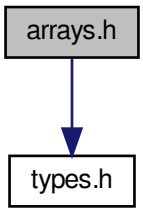
| | |
|---|---|
| *arr* | The ArrayList you want to slice, where the sliced ArrayList is saved |
| *begin* | The index of the beginning of the slice |
| *end* | The index of the end of the slice |

## 5.2 arrays.h File Reference

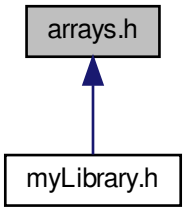
Common tasks with arrays: sorting, searching, printing etc.

```
#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 n↩ Columns)

  *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)

## 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**

| | |
|---|---|
| *spec* | Type specifier of the array to be sorted. Refer to spec_t for supported types. |
| *arr* | Pointer to the first element of the array to be sorted |
| *size* | 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**

| | |
|---|---|
| *spec* | Type specifier of the array to be sorted. Refer to spec_t for supported types |
| *arr* | Pointer to the first element of the array to be inspected |
| *size* | Number of elements of the array to be inspected |
| *...* | 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 occurence of the key in the array or the return code of the function

**Return values**

| | |
|---|---|
| *KEY_NOT_FOUND* | 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**

| | |
|---|---|
| *spec* | Type specifier of the array to be sorted. Refer to spec_t for supported types |
| *arr* | Pointer to the first element of the array to be sorted |
| *size* | 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**

| | |
|---|---|
| *spec* | Type and format specifier used to print a cell. The printf() identifier formatting convention is supported. See spec_t 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**

| | |
|---|---|
| *matrix* | Pointer to the first element of the matrix |
| *nRows* | Number of rows of the matrix |
| *nColumns* | 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 )
```

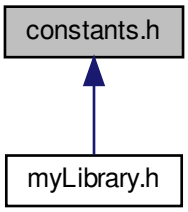
Quicksort for arrays of pointers.

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 grater 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)

### 5.3.2 Macro Definition Documentation

#### 5.3.2.1 EQUAL

```
#define EQUAL 0
```

Returned by *type*Cmp() 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 *type*Cmp() functions when first argument is grater 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 *type*Cmp() functions when first argument is smaller than the second.

#### 5.3.2.6 TRUE

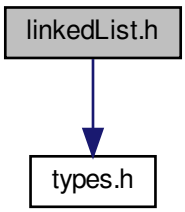
```
#define TRUE 0xFF
```

Bool value definition.

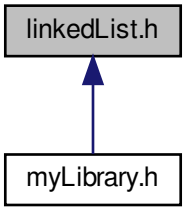
## 5.4 linkedList.h File Reference

```
#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.*
- 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.*

### 5.4.1 Function Documentation

#### 5.4.1.1 appendToLL()

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

Insert an item at the end of a LinkedList.

**Parameters**

| | |
|---|---|
| *list* | The LinkedList you want to append an item to |
| *...* | The item you want to append to `list` |

**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 newLL()

```
LinkedList newLL (
            const spec_t spec )
```

Allocate a new LinkedList of specified type.

**Parameters**

| | |
|---|---|
| *spec* | Type specifier of the LinkedList you want to create |

**Returns**

An empty LinkedList

#### 5.4.1.3 printLL()

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

Print contents from an LinkedList.

**Parameters**

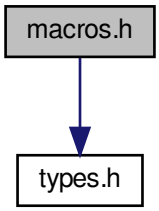
| | |
|---|---|
| *spec* | The type and format specifier you want to use to print the single element of the LinkedList |
| *list* | The LinkedList you want to print |

## 5.5 macros.h File Reference

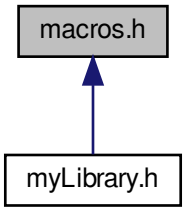
Macros for emulated overloading.

```
#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`)

**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**

| | |
|---|---|
| *arr* | Pointer to the array to be sorted |
| *size* | 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 *type*Cmp() function.

**Note**

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

**Parameters**

| | |
|---|---|
| *a* | First value to be compared |
| *b* | Second value to be compared |

**Returns**

The return code of the function called

**Return values**

| | |
|---|---|
| *GREATER* | First element is grater than the second |
| *EQUAL* | First element is equal to the second |
| *SMALLER* | 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**

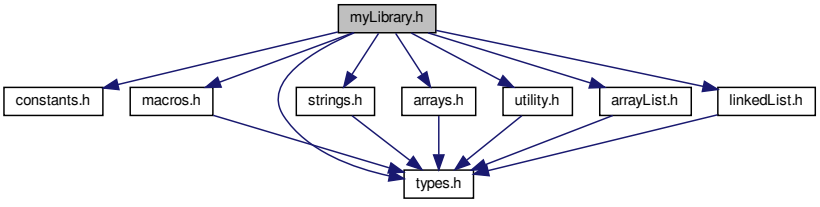
| | |
|---|---|
| *arr* | Pointer to the array to be sorted |
| *size* | Number of elements in the array to be sorted |

## 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**

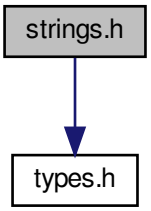
Pietro Firpo ( pietro.firpo@pm.me)

## 5.7 strings.h File Reference

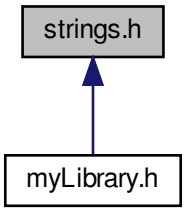
Common tasks with strings.

```
#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)

### 5.7.2 Function Documentation

#### 5.7.2.1 changeLastCharacter()

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

Get string with different last character.

**Parameters**

| | |
|---|---|
| *str* | The string you want to change the last character |
| *newCharacter* | 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**

| | |
|---|---|
| *src* | 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**

| | |
|---|---|
| *str* | The string to be inspected |
| *suffix* | The string you want to check if `string`ends with |

**Returns**

A boolean value

**Return values**

| | |
|---|---|
| *TRUE* | `str` ends with `suffix` |
| *FALSE* | `str` does not end with `suffix` |

#### 5.7.2.4 getLength()

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

Get the lenght of a string.

**Parameters**

| | |
|---|---|
| *str* | 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.

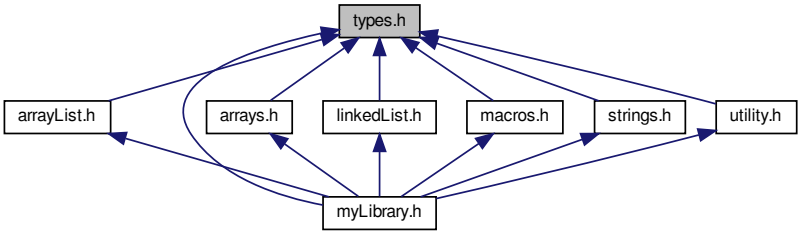
**Returns**

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)

### 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
```

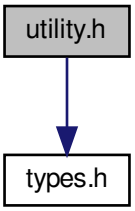
Alias for char *, used when an array of char is actually used as a string.

## 5.9 utility.h File Reference

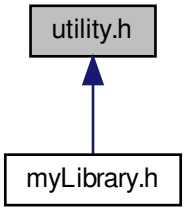
Common tasks such as comparing variables, swap bools, allocate memory.

```
#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)

## 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**

| | |
|---|---|
| *spec* | Type specifier of the values to be sorted. Refer to spec_t for supported types. |
| *a* | Pointer to the first element to be compared |
| *b* | Pointer to the second element to be compared |

**Returns**

Constant for the corresponding comparation result

**Return values**

| | |
|---|---|
| *GREATER* | First element is grater than the second |
| *EQUAL* | First element is equal to the second |
| *SMALLER* | 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**

| | |
|---|---|
| *bytes* | 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**

| | |
|---|---|
| *pointer* | Pointer to the memory to be reallocated |
| *bytes* | Number of bytes to allocate |

**Returns**

A pointer to the allocated memory

# Index