

My library

Generated by Doxygen 1.9.1

1 myLibrary homepage	1
1.1 Hi!	1
2 Data Structure Index	3
2.1 Data Structures	3
3 File Index	5
3.1 File List	5
4 Data Structure Documentation	7
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	11
4.4 Queue Struct Reference	11
4.4.1 Detailed Description	11
4.4.2 Field Documentation	12
4.4.2.1 head	12
4.4.2.2 size	12
4.4.2.3 tail	12
4.4.2.4 type	12
4.5 Stack Struct Reference	13
4.5.1 Detailed Description	13
4.5.2 Field Documentation	13
4.5.2.1 head	13
4.5.2.2 type	14
5 File Documentation	15
5.1 arrayList.h File Reference	15
5.1.1 Detailed Description	17

5.1.2 Function Documentation	17
5.1.2.1 appendToAL()	17
5.1.2.2 areALEqual()	17
5.1.2.3 bubbleSortAL()	18
5.1.2.4 chooseNewALFromArray()	18
5.1.2.5 deleteAL()	18
5.1.2.6 getALLength()	19
5.1.2.7 getFromAL()	19
5.1.2.8 insertToAL()	19
5.1.2.9 isEmpty()	21
5.1.2.10 isInAL()	21
5.1.2.11 linearSearchAL()	22
5.1.2.12 mergeAL()	22
5.1.2.13 newAL()	23
5.1.2.14 newALFromAL()	23
5.1.2.15 newALFromByteArray()	23
5.1.2.16 newALFromCharArray()	24
5.1.2.17 newALFromDoubleArray()	24
5.1.2.18 newALFromFloatArray()	24
5.1.2.19 newALFromIntArray()	24
5.1.2.20 newALFromPtrArray()	24
5.1.2.21 printAL()	24
5.1.2.22 quickSortAL()	25
5.1.2.23 removeFromAL()	25
5.1.2.24 reverseAL()	25
5.1.2.25 setALItem()	26
5.1.2.26 sliceAL()	26
5.2 arrays.h File Reference	26
5.2.1 Detailed Description	28
5.2.2 Function Documentation	28
5.2.2.1 charBubbleSort()	28
5.2.2.2 charLinearSearch()	29
5.2.2.3 charQuickSort()	29
5.2.2.4 chooseBubbleSortArr()	29
5.2.2.5 chooseLinearSearch()	29
5.2.2.6 chooseQuickSortArr()	30
5.2.2.7 doubleBubbleSort()	30
5.2.2.8 doubleLinearSearch()	31
5.2.2.9 doubleQuickSort()	31
5.2.2.10 floatBubbleSort()	31
5.2.2.11 floatLinearSearch()	31
5.2.2.12 floatQuickSort()	31

5.2.2.13 intBubbleSort()	32
5.2.2.14 intLinearSearch()	32
5.2.2.15 intQuickSort()	32
5.2.2.16 printMatrix()	32
5.2.2.17 ptrBubbleSort()	33
5.2.2.18 ptrLinearSearch()	33
5.2.2.19 ptrQuickSort()	33
5.3 constants.h File Reference	33
5.3.1 Detailed Description	34
5.3.2 Macro Definition Documentation	34
5.3.2.1 EQUAL	34
5.3.2.2 FALSE	34
5.3.2.3 GREATER	34
5.3.2.4 KEY_NOT_FOUND	35
5.3.2.5 SMALLER	35
5.3.2.6 TRUE	35
5.4 linkedList.h File Reference	35
5.4.1 Detailed Description	37
5.4.2 Function Documentation	37
5.4.2.1 appendToLL()	37
5.4.2.2 appendToLLFromPtr()	37
5.4.2.3 areLLEqual()	38
5.4.2.4 chooseNewLLFromArray()	38
5.4.2.5 deleteLL()	38
5.4.2.6 getFromLL()	39
5.4.2.7 getLLLength()	39
5.4.2.8 insertToLL()	39
5.4.2.9 isInLL()	40
5.4.2.10 isLLEmpty()	40
5.4.2.11 linearSearchLL()	41
5.4.2.12 linearSearchLLPtr()	41
5.4.2.13 mergeLL()	42
5.4.2.14 newLL()	42
5.4.2.15 newLLFromCharArray()	43
5.4.2.16 newLLFromDoubleArray()	43
5.4.2.17 newLLFromFloatArray()	43
5.4.2.18 newLLFromIntArray()	43
5.4.2.19 newLLFromLL()	43
5.4.2.20 newLLFromPtrArray()	44
5.4.2.21 printLL()	44
5.4.2.22 removeFromLL()	44
5.4.2.23 setLLItem()	45

5.4.2.24 sliceLL()	45
5.5 macros.h File Reference	45
5.5.1 Detailed Description	47
5.5.2 Macro Definition Documentation	48
5.5.2.1 append	48
5.5.2.2 areEqual	48
5.5.2.3 bubbleSortArr	49
5.5.2.4 cmpVal	49
5.5.2.5 delete	50
5.5.2.6 deleteHead	50
5.5.2.7 getHeadData	50
5.5.2.8 getItem	51
5.5.2.9 getLength	51
5.5.2.10 insert	52
5.5.2.11 isEmpty	52
5.5.2.12 isIn	53
5.5.2.13 linearSearch	53
5.5.2.14 merge	54
5.5.2.15 newALFromArray	54
5.5.2.16 newLLFromArray	55
5.5.2.17 newQueueFromArray	55
5.5.2.18 newStackFromArray [1/2]	56
5.5.2.19 newStackFromArray [2/2]	56
5.5.2.20 print	57
5.5.2.21 quickSortArr	57
5.5.2.22 removeItem	58
5.5.2.23 set	58
5.5.2.24 slice	59
5.6 myLibrary.h File Reference	59
5.6.1 Detailed Description	60
5.7 queue.h File Reference	60
5.7.1 Detailed Description	61
5.7.2 Function Documentation	61
5.7.2.1 areQueuesEqual()	61
5.7.2.2 chooseNewQueueFromArray()	62
5.7.2.3 deleteHeadFromQueue()	62
5.7.2.4 deleteQueue()	63
5.7.2.5 dequeue()	63
5.7.2.6 enqueue()	63
5.7.2.7 enqueueFromPtr()	64
5.7.2.8 getHeadDataFromQueue()	64
5.7.2.9 getQueueLength()	64

5.7.2.10 isInQueue()	65
5.7.2.11 isQueueEmpty()	65
5.7.2.12 newQueue()	65
5.7.2.13 newQueueFromCharArray()	66
5.7.2.14 newQueueFromDoubleArray()	66
5.7.2.15 newQueueFromFloatArray()	66
5.7.2.16 newQueueFromIntArray()	67
5.7.2.17 newQueueFromPtrArray()	67
5.7.2.18 printQueue()	67
5.8 stack.h File Reference	67
5.8.1 Detailed Description	69
5.8.2 Function Documentation	69
5.8.2.1 areStacksEqual()	69
5.8.2.2 chooseNewStackFromArray()	70
5.8.2.3 deleteHeadFromStack()	70
5.8.2.4 deleteStack()	70
5.8.2.5 getHeadDataFromStack()	71
5.8.2.6 getStackLength()	71
5.8.2.7 isInStack()	71
5.8.2.8 isStackEmpty()	72
5.8.2.9 newStack()	72
5.8.2.10 newStackFromCharArray()	73
5.8.2.11 newStackFromDoubleArray()	73
5.8.2.12 newStackFromFloatArray()	73
5.8.2.13 newStackFromIntArray()	73
5.8.2.14 newStackFromPtrArray()	74
5.8.2.15 pop()	74
5.8.2.16 printStack()	74
5.8.2.17 push()	74
5.8.2.18 pushFromPtr()	75
5.9 strings.h File Reference	75
5.9.1 Detailed Description	76
5.9.2 Function Documentation	76
5.9.2.1 changeLastCharacter()	76
5.9.2.2 copyOf()	77
5.9.2.3 endsWith()	77
5.9.2.4 getString()	78
5.10 types.h File Reference	78
5.10.1 Detailed Description	79
5.10.2 Typedef Documentation	79
5.10.2.1 byte	79
5.10.2.2 Node	79

5.10.2.3 spec_t	80
5.10.2.4 string	80
5.11 utility.h File Reference	80
5.11.1 Detailed Description	81
5.11.2 Function Documentation	81
5.11.2.1 byteCmp()	81
5.11.2.2 charCmp()	81
5.11.2.3 chooseCmp()	82
5.11.2.4 doubleCmp()	82
5.11.2.5 floatCmp()	82
5.11.2.6 intCmp()	83
5.11.2.7 ptrCmp()	83
5.11.2.8 saferMalloc()	83
5.11.2.9 saferRealloc()	83
Index	85

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:

ArrayList		
	ArrayList type	7
LinkedList		
	LinkedList type	8
node		
	Node type	10
Queue		
	Queue type	11
Stack		
	Stack type	13

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

arrayList.h	Functions for working with ArrayList type	15
arrays.h	Common tasks with arrays: sorting, searching, printing etc	26
constants.h	Definition of symbolic constants used by the library	33
linkedList.h	Functions for working with LinkedList type	35
macros.h	Macros for emulated overloading	45
myLibrary.h	Includes all other headers. Useful for rapid import	59
queue.h	Functions for working with Queue type	60
stack.h	Functions for working with Stack type	67
strings.h	Common tasks with strings	75
types.h	Collection of useful types	78
utility.h	Common tasks such as comparing variables, allocate memory	80

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:

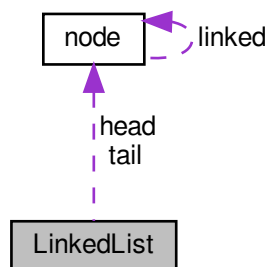
- [types.h](#)

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:

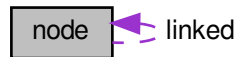
- [types.h](#)

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:

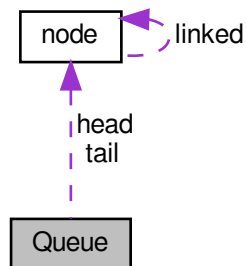
- [types.h](#)

4.4 Queue Struct Reference

[Queue](#) type

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

Collaboration diagram for [Queue](#):



Data Fields

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

4.4.1 Detailed Description

[Queue](#) type

Note

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

4.4.2 Field Documentation

4.4.2.1 head

`Node Queue::head`

Head of the [Queue](#).

4.4.2.2 size

`unsigned int Queue::size`

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

4.4.2.3 tail

`Node Queue::tail`

Tail of the [Queue](#).

4.4.2.4 type

`spec_t Queue::type`

The type of the elements contained by the [Queue](#). Refer to [spec_t](#).

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

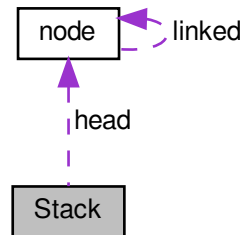
- [types.h](#)

4.5 Stack Struct Reference

[Stack](#) type

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

Collaboration diagram for Stack:



Data Fields

- [spec_t](#) type
The type of the elements contained by the [Stack](#). Refer to [spec_t](#).
- [Node](#) head
Head of the [Stack](#).

4.5.1 Detailed Description

[Stack](#) type

Note

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

4.5.2 Field Documentation

4.5.2.1 head

[Node](#) `Stack::head`

Head of the [Stack](#).

4.5.2.2 type

`spec_t` `Stack::type`

The type of the elements contained by the [Stack](#). Refer to [spec_t](#).

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

- [types.h](#)

Chapter 5

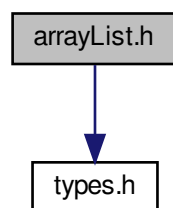
File Documentation

5.1 arrayList.h File Reference

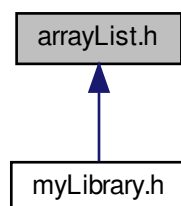
Functions for working with [ArrayList](#) type.

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

Include dependency graph for arrayList.h:



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



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 item at a specified position of an [ArrayList](#).
- void [setALItem](#) ([ArrayList](#) list, unsigned int index,...)
Set value of an item 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 item 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 a static array.
- [ArrayList newALFromCharArray](#) (const char list[], unsigned int size)
Create [ArrayList](#) from a list of chars.
- [ArrayList newALFromByteArray](#) (const char list[], unsigned int size)
Create [ArrayList](#) from a list of bytes.
- [ArrayList newALFromIntArray](#) (const int list[], unsigned int size)
Create [ArrayList](#) from a list of ints.
- [ArrayList newALFromFloatArray](#) (const float list[], unsigned int size)
Create [ArrayList](#) from a 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.
- unsigned int [getALLength](#) (const [ArrayList](#) list)
Get the size of an [ArrayList](#).
- [byte isALEmpty](#) ([ArrayList](#) list)
Check if [ArrayList](#) is empty.

5.1.1 Detailed Description

Functions for working with `ArrayList` type.

Author

Pietro Firpo (pietro.firpo@pm.me)

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 <code>ArrayList</code> 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 for single call 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 <code>ArrayList</code> you want to compare
<i>list2</i>	The second <code>ArrayList</code> 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 ArrayList 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 a static array.

Parameters

<i>spec</i>	The type specifier of the array passed. Refer to <code>spec_t</code>
<i>list</i>	The list you want to create the ArrayList from
<i>size</i>	The number of items in <code>list</code>

Returns

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

5.1.2.5 deleteAL()

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

Delete an [ArrayList](#).

Parameters

<i>list</i>	The ArrayList you want to delete
-------------	--

5.1.2.6 getALLength()

```
unsigned int getALLength (
    const ArrayList list )
```

Get the size of an [ArrayList](#).

Parameters

<i>list</i>	The ArrayList you want to evaluate
-------------	--

Returns

The number of items in `list`

5.1.2.7 getFromAL()

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

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

Parameters

<i>list</i>	The ArrayList 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

5.1.2.8 insertToAL()

```
void insertToAL (
    ArrayList list,
```

```
    unsigned int index,  
    ... )
```

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

Parameters

<i>list</i>	The ArrayList you want to insert an item into
<i>index</i>	The position you want to insert an item at
...	The item you want to insert into <i>list</i>

Note

Even though inserting more than one item for single call 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

5.1.2.9 isALEmpty()

```
byte isALEmpty (
    ArrayList list )
```

Check if [ArrayList](#) is empty.

Parameters

<i>list</i>	The ArrayList to be checked
-------------	---

Return values

<i>TRUE</i>	<i>list</i> is empty
<i>FALSE</i>	<i>list</i> is not empty

5.1.2.10 isInAL()

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

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

Parameters

<i>list</i>	The ArrayList you want search in
...	The item you want to search

Note

Even though searching more than one item for single call 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 item is contained in <code>list</code>
<i>FALSE</i>	Given item is not contained in <code>list</code>

5.1.2.11 linearSearchAL()

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

Linear search for [ArrayList](#).

Parameters

<i>list</i>	The ArrayList to be inspected
<i>...</i>	The key to be searched

Note

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

Even though passing more than one key does not throw a compiler nor runtime error, only searching one key 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
----------------------	-----------------------

5.1.2.12 mergeAL()

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

Merge two [ArrayList](#).

Parameters

<i>list1</i>	The first ArrayList to be merged, where the merged ArrayList is saved
<i>list2</i>	The second ArrayList to be merged

5.1.2.13 newAL()

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

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

Parameters

<i>spec</i>	Type specifier of the ArrayList you want to create
-------------	--

Returns

An empty [ArrayList](#)

5.1.2.14 newALFromAL()

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

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

Parameters

<i>list</i>	The ArrayList you want to copy
-------------	--

Returns

A copy of `list`

5.1.2.15 newALFromByteArray()

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

Create [ArrayList](#) from a list of bytes.

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

5.1.2.16 newALFromCharArray()

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

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

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

5.1.2.17 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\(\)](#)

5.1.2.18 newALFromFloatArray()

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

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

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

5.1.2.19 newALFromIntArray()

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

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

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

5.1.2.20 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\(\)](#)

5.1.2.21 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 item of the ArrayList . Use the <code>printf()</code> conventions
<i>list</i>	The ArrayList you want to print

5.1.2.22 quickSortAL()

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

Quicksort for [ArrayList](#).

Parameters

<i>list</i>	The ArrayList you want to quicksort
-------------	---

5.1.2.23 removeFromAL()

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

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

Parameters

<i>list</i>	The ArrayList you want to delete an item from
<i>index</i>	The index of the item you want to delete

5.1.2.24 reverseAL()

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

Reverse an [ArrayList](#).

Parameters

<i>list</i>	The ArrayList you want to reverse
-------------	---

5.1.2.25 setALItem()

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

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

Parameters

<i>list</i>	The ArrayList you want to edit
<i>index</i>	The index of the item you want to change
<i>...</i>	The item you want to set the index-th item of <i>list</i> to

Note

Even though changing more than one item for single call 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

5.1.2.26 sliceAL()

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

Slice an [ArrayList](#).

Parameters

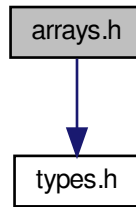
<i>list</i>	The ArrayList you want to slice, where the sliced ArrayList is saved
<i>begin</i>	The index of the beginning of the slice
<i>end</i>	The index of the end of the slice

5.2 arrays.h File Reference

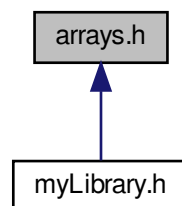
Common tasks with arrays: sorting, searching, printing etc.

```
#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 a 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.*
- int [charLinearSearch](#) (const char *arr, int size, char key)
- Linear search for arrays of chars.*
- int [intLinearSearch](#) (const char *arr, int size, int key)
- Linear search for arrays of integers.*
- int [floatLinearSearch](#) (const char *arr, int size, float key)
- Linear search for arrays of floats.*
- int [doubleLinearSearch](#) (const char *arr, int size, double key)
- Linear search for arrays of doubles.*
- int [ptrLinearSearch](#) (const char **arr, int size, void *key)
- Linear search 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 charLinearSearch()

```
int charLinearSearch (
    const char * arr,
    int size,
    char key )
```

Linear search for arrays of chars.

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

5.2.2.3 charQuickSort()

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

Quicksort for arrays of chars.

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

5.2.2.4 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 spec_t 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

5.2.2.5 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 spec_t 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 key is supported. Other items are ignored. If you don't specify any key 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
----------------------	-----------------------

5.2.2.6 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 spec_t 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

5.2.2.7 doubleBubbleSort()

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

Bubblesort for arrays of doubles.

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

5.2.2.8 doubleLinearSearch()

```
int doubleLinearSearch (
    const char * arr,
    int size,
    double key )
```

Linear search for arrays of doubles.

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

5.2.2.9 doubleQuickSort()

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

Quicksort for arrays of doubles.

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

5.2.2.10 floatBubbleSort()

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

Bubblesort for arrays of floats.

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

5.2.2.11 floatLinearSearch()

```
int floatLinearSearch (
    const char * arr,
    int size,
    float key )
```

Linear search for arrays of floats.

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

5.2.2.12 floatQuickSort()

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

Quicksort for arrays of floats.

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

5.2.2.13 intBubbleSort()

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

Bubblesort for arrays of ints.

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

5.2.2.14 intLinearSearch()

```
int intLinearSearch (
    const char * arr,
    int size,
    int key )
```

Linear search for arrays of integers.

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

5.2.2.15 intQuickSort()

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

Quicksort for arrays of ints.

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

5.2.2.16 printMatrix()

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

Print a 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 and 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

<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

5.2.2.17 ptrBubbleSort()

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

Bubblesort for arrays of pointers.

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

5.2.2.18 ptrLinearSearch()

```
int ptrLinearSearch (
    const char ** arr,
    int size,
    void * key )
```

Linear search for arrays of pointers.

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

5.2.2.19 ptrQuickSort()

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

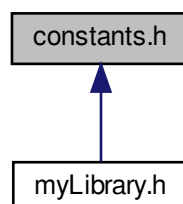
Quicksort for arrays of pointers.

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)

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
```

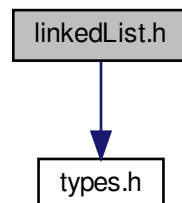
Bool value definition.

5.4 linkedList.h File Reference

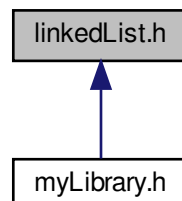
Functions for working with [LinkedList](#) type.

```
#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 a *LinkedList*.
- void `appendToLL` (*LinkedList* list,...)
Insert an item at the end of a *LinkedList*.
- void `appendToLLFromPtr` (*LinkedList* list, const 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 a *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*.
- unsigned int `getLLLength` (const *LinkedList* list)
Get the size of a *LinkedList*.
- `LinkedList newLLFromCharArray` (const char arr[], unsigned int size)
Create a *LinkedList* from a array of chars.
- `LinkedList newLLFromIntArray` (const int arr[], unsigned int size)
Create a *LinkedList* from a array of ints.
- `LinkedList newLLFromFloatArray` (const float arr[], unsigned int size)
Create a *LinkedList* from a array of floats.
- `LinkedList newLLFromDoubleArray` (const double arr[], unsigned int size)
Create a *LinkedList* from an array of doubles.
- `LinkedList newLLFromPtrArray` (const void *arr, unsigned int size)
Create a *LinkedList* from an array of pointers.
- `byte isLLEmpty` (*LinkedList* list)
Check if *LinkedList* is empty.

5.4.1 Detailed Description

Functions for working with `LinkedList` type.

Author

Pietro Firpo (pietro.firpo@pm.me)

5.4.2 Function Documentation

5.4.2.1 `appendToLL()`

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

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

Parameters

<i>list</i>	The <code>LinkedList</code> 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 for single call 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 `LinkedList` can be messed up

5.4.2.2 `appendToLLFromPtr()`

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

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

Parameters

<i>list</i>	The <code>LinkedList</code> you want to append an item to
<i>element</i>	Pointer to the item you want to append to <code>list</code>

5.4.2.3 areLLEqual()

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

Compare two [LinkedList](#).

Parameters

<i>list1</i>	The first LinkedList you want to compare
<i>list2</i>	The second LinkedList 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

5.4.2.4 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 spec_t for supported types
<i>arr</i>	The array you want to create the LinkedList from
<i>size</i>	The number of items of <code>list</code>

Returns

A [LinkedList](#) containing the elements in `list` in the same order

5.4.2.5 deleteLL()

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

Delete a [LinkedList](#).

Parameters

<i>list</i>	The LinkedList you want to delete
-------------	---

5.4.2.6 `getFromLL()`

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

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

Parameters

<i>list</i>	The LinkedList 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

5.4.2.7 `getLLLength()`

```
unsigned int getLLLength (
    const LinkedList list )
```

Get the size of a [LinkedList](#).

Parameters

<i>list</i>	The LinkedList you want to evaluate
-------------	---

Returns

The number of elements in `list`

5.4.2.8 `insertToLL()`

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

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

Parameters

<i>list</i>	The LinkedList you want to insert an element into
<i>index</i>	The position you want to insert an element at
...	The item you want to insert into <code>list</code>

Note

Even though inserting more than one item for single call 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 [LinkedList](#) can be messed up

5.4.2.9 `isInLL()`

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

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

Parameters

<i>list</i>	The LinkedList you want search in
...	The element you want to search

Note

Even though checking more than one item for single call does not throw a compiler nor runtime error, only checking one item is supported. Other items are ignored. If you don't specify any item to be checked, 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>

5.4.2.10 `isLLEmpty()`

```
byte isLLEmpty (
    LinkedList list )
```

Check if [LinkedList](#) is empty.

Parameters

<i>list</i>	The LinkedList to be checked
-------------	--

Return values

<i>TRUE</i>	<i>list</i> is empty
<i>FALSE</i>	<i>list</i> is not empty

5.4.2.11 linearSearchLL()

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

Linear search for [LinkedList](#).

Parameters

<i>list</i>	The LinkedList to be inspected
<i>...</i>	The key to be searched

Note

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

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
----------------------	-----------------------

5.4.2.12 linearSearchLLPtr()

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

Linear search for [LinkedList](#).

Parameters

<i>list</i>	The LinkedList to be inspected
<i>...</i>	The key to be searched

Note

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

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

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

5.4.2.13 mergeLL()

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

Merge two [LinkedList](#).

Parameters

<i>list1</i>	The first LinkedList to be merged, where the merged LinkedList is saved
<i>list2</i>	The second LinkedList to be merged

5.4.2.14 newLL()

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

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

Parameters

<i>spec</i>	Type specifier of the LinkedList you want to create. Refer to spec_t for supported types
-------------	--

Returns

An empty [LinkedList](#)

5.4.2.15 `newLLFromCharArray()`

```
LinkedList newLLFromCharArray (
    const char arr[],
    unsigned int size )
```

Create a [LinkedList](#) from a array of chars.

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

5.4.2.16 `newLLFromDoubleArray()`

```
LinkedList newLLFromDoubleArray (
    const double arr[],
    unsigned int size )
```

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

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

5.4.2.17 `newLLFromFloatArray()`

```
LinkedList newLLFromFloatArray (
    const float arr[],
    unsigned int size )
```

Create a [LinkedList](#) from a array of floats.

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

5.4.2.18 `newLLFromIntArray()`

```
LinkedList newLLFromIntArray (
    const int arr[],
    unsigned int size )
```

Create a [LinkedList](#) from a array of ints.

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

5.4.2.19 `newLLFromLL()`

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

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

Parameters

<i>list</i>	The LinkedList you want to copy
-------------	---

Returns

A copy of `list`

5.4.2.20 newLLFromPtrArray()

```
LinkedList newLLFromPtrArray (
    const void * arr,
    unsigned int size )
```

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

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

5.4.2.21 printLL()

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

Print contents from a [LinkedList](#).

Parameters

<i>spec</i>	The type and format specifier you want to use to print the single element of the LinkedList . Use the <code>printf()</code> conventions
<i>list</i>	The LinkedList you want to print

5.4.2.22 removeFromLL()

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

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

Parameters

<i>list</i>	The LinkedList you want to delete an item from
<i>index</i>	The index of the item you want to delete

5.4.2.23 setLLItem()

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

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

Parameters

<i>list</i>	The LinkedList you want to edit
<i>index</i>	The index of the element you want to change
...	The item you want to set the index-th element of <code>list</code> to

Note

Even though changing more than one item for single call 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

5.4.2.24 sliceLL()

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

Slice a [LinkedList](#).

Parameters

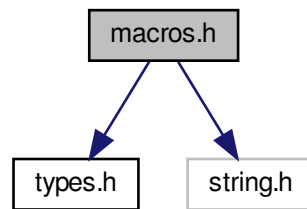
<i>list</i>	The LinkedList you want to slice, where the sliced LinkedList is saved
<i>begin</i>	The index of the beginning of the slice
<i>end</i>	The index of the end of the slice

5.5 macros.h File Reference

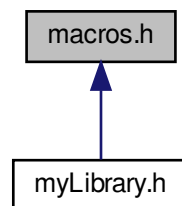
Macros for emulated overloading.

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

Include dependency graph for macros.h:



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



Macros

- #define `cmpVal(a, b)`
Compare two values.
- #define `bubbleSortArr(arr, size)`
BubbleSort for arrays.
- #define `quickSortArr(arr, size)`
Quicksort for arrays.
- #define `newALFromArray(arr, size)`
Create an `ArrayList` from a static array.
- #define `newLLFromArray(arr, size)`
Create a `LinkedList` from a static array.
- #define `newStackFromArray(arr, size)`
Create a `Stack` from a static array.
- #define `newQueueFromArray(arr, size)`
Create a `Queue` from a static array.
- #define `newStackFromArray(arr, size)`
Create a `Stack` from a static array.
- #define `print(spec, collection)`

- *Print contents from an [ArrayList](#), [LinkedList](#), [Stack](#) or [Queue](#).*
- `#define areEqual(collection1, collection2)`
Compare two [ArrayList](#), [LinkedList](#), [Stack](#) or [Queue](#).
- `#define append(list, item)`
Insert an item at the end of an [ArrayList](#) or [LinkedList](#).
- `#define insert(list, index, item)`
Insert an element at a specified position of an [ArrayList](#) or [LinkedList](#).
- `#define set(list, index, newItem)`
Set value of an element of an [ArrayList](#) or [LinkedList](#).
- `#define merge(list1, list2)`
Merge two [ArrayList](#) or [LinkedList](#).
- `#define slice(list, begin, end)`
Slice an [ArrayList](#) or [LinkedList](#).
- `#define removeItem(list, index)`
Remove an item from an [ArrayList](#) or [LinkedList](#).
- `#define getItem(list, index, dest)`
Get an item from an [ArrayList](#) or [LinkedList](#).
- `#define delete(collection)`
Delete an [ArrayList](#), [LinkedList](#), [Stack](#) or [Queue](#).
- `#define isIn(collection, item)`
Detect if an item is inside an [ArrayList](#), [LinkedList](#), [Stack](#) or [Queue](#).
- `#define getLength(collection)`
Get the number of elements in an [ArrayList](#), [LinkedList](#), [Stack](#), [Queue](#) or [string](#).
- `#define linearSearch(list, key)`
Linear search for an [ArrayList](#) or [LinkedList](#).
- `#define deleteHead(collection)`
Delete current [Stack](#) or [Queue](#) head.
- `#define isEmpty(collection)`
Check if an [ArrayList](#), [LinkedList](#), [Stack](#) or [Queue](#) is empty.
- `#define getHeadData(collection, dest)`
Get the item at the head of a [Stack](#) or [Queue](#) without popping/dequeueing it.

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 these error reportings

5.5.2 Macro Definition Documentation

5.5.2.1 append

```
#define append(  
    list,  
    item )
```

Value:

```
_Generic(list, ArrayList \  
: appendToAL, LinkedList \  
: appendToLL)(list, item)
```

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

Parameters

<i>list</i>	The list you want to append an item to
<i>item</i>	The item you want to append to <i>list</i>

5.5.2.2 areEqual

```
#define areEqual(  
    collection1,  
    collection2 )
```

Value:

```
_Generic(collection1, ArrayList \  
: areALEqual, LinkedList \  
: areLLEqual, Stack \  
: areStacksEqual, Queue \  
: areQueuesEqual)(collection1, collection2)
```

Compare two [ArrayList](#), [LinkedList](#), [Stack](#) or [Queue](#).

Parameters

<i>collection1</i>	The first ArrayList , LinkedList , Stack or Queue you want to compare
<i>collection2</i>	The second ArrayList , LinkedList , Stack or Queue you want to compare

Note

Passing two different types (for example, an [ArrayList](#) and a [Stack](#)) does not throw errors but does not work and the result can be unpredictable

5.5.2.3 bubbleSortArr

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

Value:

```
_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

5.5.2.4 cmpVal

```
#define cmpVal(  
    a,  
    b )
```

Value:

```
_Generic((a, b), char *      \  
: charCmp, int *            \  
: intCmp, float *          \  
: floatCmp, double *       \  
: doubleCmp, void **       \  
: ptrCmp)(a, b)
```

Compare two values.

Parameters

<i>a</i>	Pointer to the first value to be compared
<i>b</i>	Pointer to the 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
----------------	---

Return values

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

5.5.2.5 delete

```
#define delete(
    collection )
```

Value:

```
_Generic(collection, ArrayList \
: deleteAL, LinkedList \
: deleteLL, Stack \
: deleteStack, Queue \
: deleteQueue)(collection)
```

Delete an [ArrayList](#), [LinkedList](#), [Stack](#) or [Queue](#).

Parameters

<i>collection</i>	The ArrayList , LinkedList , Stack or Queue you want to delete
-------------------	--

5.5.2.6 deleteHead

```
#define deleteHead(
    collection )
```

Value:

```
_Generic(list, Stack \
: deleteHeadFromStack, Queue \
: deleteHeadFromQueue)(collection)
```

Delete current [Stack](#) or [Queue](#) head.

Parameters

<i>stack</i>	The Stack or Queue you want to delete the head from
--------------	---

5.5.2.7 getHeadData

```
#define getHeadData(
    collection,
    dest )
```

Value:

```
_Generic(list, Stack
: getHeadDataFromStack, Queue \
: getHeadDataFromQueue) (collection)
```

Get the item at the head of a [Stack](#) or [Queue](#) without popping/dequeueing it.

Parameters

<i>collection</i>	The Stack or Queue you want to get the item from
<i>dest</i>	The address of the variable you want to store the item in

5.5.2.8 getItem

```
#define getItem(
    list,
    index,
    dest )
```

Value:

```
_Generic(list, ArrayList
: getFromAL, LinkedList \
: getFromLL) (list, index, dest)
```

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

Parameters

<i>list</i>	The list 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

5.5.2.9 getLength

```
#define getLength(
    collection )
```

Value:

```
_Generic(collection, ArrayList
: getALLength, LinkedList \
: getLLLength, Stack \
: getStackLength, Queue \
: getQueueLength, string \
: strlen) (collection)
```

Get the number of elements in an [ArrayList](#), [LinkedList](#), [Stack](#), [Queue](#) or [string](#).

Parameters

<i>collection</i>	The ArrayList , LinkedList , Stack , Queue or string you want to evaluate
-------------------	---

Returns

The number of elements in `collection`

5.5.2.10 insert

```
#define insert(
    list,
    index,
    item )
```

Value:

```
_Generic(list, ArrayList \
: insertToAL, LinkedList \
: insertToLL)(list, index, item)
```

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

Parameters

<i>list</i>	The list you want to insert an element into
<i>index</i>	The position you want to insert an item at
<i>item</i>	The item you want to insert into <code>list</code>

5.5.2.11 isEmpty

```
#define isEmpty(
    collection )
```

Value:

```
_Generic(collection, ArrayList \
: isEmpty, LinkedList \
: isEmpty, Stack \
: isEmpty, Queue \
: isEmpty)(collection, item)
```

Check if an [ArrayList](#), [LinkedList](#), [Stack](#) or [Queue](#) is empty.

Parameters

<i>collection</i>	The ArrayList , LinkedList , Stack or Queue to be checked
-------------------	---

Return values

<i>TRUE</i>	<code>collection</code> is empty
<i>FALSE</i>	<code>collection</code> is not empty

5.5.2.12 isIn

```
#define isIn(
    collection,
    item )
```

Value:

```
_Generic(collection, ArrayList \
: isInAL, LinkedList \
: isInLL, Stack \
: isInStack, Queue \
: isInQueue)(collection, item)
```

Detect if an item is inside an [ArrayList](#), [LinkedList](#), [Stack](#) or [Queue](#).

Parameters

<i>collection</i>	The ArrayList , LinkedList , Stack or Queue you want search in
<i>item</i>	The item you want to search

Note

Passing float or double [ArrayList](#), [LinkedList](#), [Stack](#) or [Queue](#) is not supported

Return values

<i>TRUE</i>	Given item is contained in <i>collection</i>
<i>FALSE</i>	Given item is not contained in <i>collection</i>

5.5.2.13 linearSearch

```
#define linearSearch(
    list,
    key )
```

Value:

```
_Generic(list, ArrayList \
: linearSearchAL, LinkedList \
: linearSearchLL)(list, key)
```

Linear search for an [ArrayList](#) or [LinkedList](#).

Parameters

<i>list</i>	The ArrayList or LinkedList to be inspected
<i>key</i>	The key to be searched

Note

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

Returns

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

Return values

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

5.5.2.14 merge

```
#define merge(
    list1,
    list2 )
```

Value:

```
_Generic(list1, ArrayList \
: mergeAL, LinkedList \
: mergeLL)(list1, list2)
```

Merge two [ArrayList](#) or [LinkedList](#).

Parameters

<i>list1</i>	The first list to be merged, where the merged list is saved
<i>list2</i>	The second list to be merged

Note

Passing an [ArrayList](#) and a [LinkedList](#) does not throw errors but does not work and `list1` is messed up

5.5.2.15 newALFromArray

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

Value:

```
_Generic(arr, char * \
: newALFromCharArray, int * \
: newALFromArray, float * \
: newALFromArray, double * \
: newALFromArray, void ** \
: newALFromArray)(arr, size)
```

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

Parameters

<i>arr</i>	The array you want to create an ArrayList from
<i>size</i>	The size of <code>arr</code>

Returns

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

5.5.2.16 newLLFromArray

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

Value:

```
_Generic(arr, char *  
: newLLFromCharArray, int *      \  
: newLLFromIntArray, float *    \  
: newLLFromFloatArray, double * \  
: newLLFromDoubleArray, void ** \  
: newLLFromPtrArray)(arr, size)
```

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

Parameters

<i>arr</i>	The array you want to create a LinkedList from
<i>size</i>	The size of <code>arr</code>

Returns

A [LinkedList](#) containing all the elements of `arr` in the same order

5.5.2.17 newQueueFromArray

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

Value:

```
_Generic(arr, char *  
: newQueueFromCharArray, int *      \  
: newQueueFromIntArray, float *    \  
: newQueueFromFloatArray, double * \  
: newQueueFromDoubleArray, void ** \  
: newQueueFromPtrArray)(arr, size)
```

Create a [Queue](#) from a static array.

Parameters

<i>arr</i>	The array you want to create a Queue from
<i>size</i>	The size of <code>arr</code>

Returns

A [Queue](#) containing all the elements of `arr` with the first element of `arr` as head

5.5.2.18 newStackFromArray [1/2]

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

Value:

```
_Generic(arr, char *
: newStackFromCharArray, int * \
: newStackFromIntArray, float * \
: newStackFromFloatArray, double * \
: newStackFromDoubleArray, void ** \
: newStackFromPtrArray)(arr, size)
```

Create a [Stack](#) from a static array.

Parameters

<i>arr</i>	The array you want to create a Stack from
<i>size</i>	The size of <code>arr</code>

Returns

A [Stack](#) containing all the elements of `arr` with the last element of `arr` as head

Parameters

<i>arr</i>	The array you want to create a Stack from
<i>size</i>	The size of <code>arr</code>

Returns

A [Stack](#) containing all the elements of `arr` with the first element of `arr` as head

5.5.2.19 newStackFromArray [2/2]

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

Value:

```
_Generic(arr, char *
: newStackFromCharArray, int * \
: newStackFromIntArray, float * \
: newStackFromFloatArray, double * \
: newStackFromDoubleArray, void ** \
: newStackFromPtrArray)(arr, size)
```

Create a [Stack](#) from a static array.

Parameters

<i>arr</i>	The array you want to create a Stack from
<i>size</i>	The size of <i>arr</i>

Returns

A [Stack](#) containing all the elements of *arr* with the last element of *arr* as head

Parameters

<i>arr</i>	The array you want to create a Stack from
<i>size</i>	The size of <i>arr</i>

Returns

A [Stack](#) containing all the elements of *arr* with the first element of *arr* as head

5.5.2.20 print

```
#define print(  
    spec,  
    collection )
```

Value:

```
_Generic(collection, ArrayList      \  
: printAL, LinkedList \  
: printLL, Stack      \  
: printStack, Queue   \  
: printQueue)(spec, collection)
```

Print contents from an [ArrayList](#), [LinkedList](#), [Stack](#) or [Queue](#).

Parameters

<i>spec</i>	The type and format specifier you want to use to print the single element. Use the <code>printf()</code> conventions
<i>collection</i>	The ArrayList , LinkedList , Stack or Queue you want to print

5.5.2.21 quickSortArr

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

Value:

```

_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

5.5.2.22 removeItem

```

#define removeItem(
    list,
    index )

```

Value:

```

_Generic(list, ArrayList \
: removeFromAL, LinkedList \
: removeFromLL)(list, index)

```

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

Parameters

<i>list</i>	The list you want to delete an item from
<i>index</i>	The index of the item you want to delete

5.5.2.23 set

```

#define set(
    list,
    index,
    newItem )

```

Value:

```

_Generic(list, ArrayList \
: setALItem, LinkedList \
: setLLItem)(list, index, newItem)

```

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

Parameters

<i>list</i>	The list you want to edit
<i>index</i>	The index of the item you want to change
<i>newItem</i>	The item you want to set the index-th element of <code>list</code> to

5.5.2.24 slice

```
#define slice(
    list,
    begin,
    end )
```

Value:

```
_Generic(list, ArrayList \
: sliceAL, LinkedList \
: sliceLL)(list, begin, end)
```

Slice an [ArrayList](#) or [LinkedList](#).

Parameters

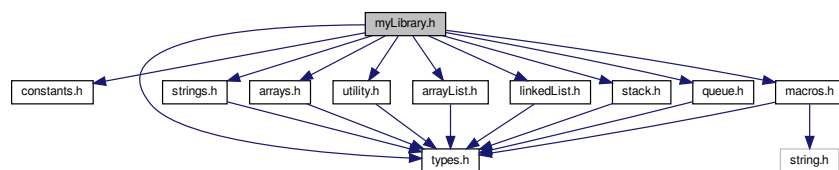
<i>list</i>	The list you want to slice, where the sliced list is saved
<i>begin</i>	The index of the beginning of the slice
<i>end</i>	The index of the end of the slice

5.6 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 "stack.h"
#include "queue.h"
```

Include dependency graph for myLibrary.h:



5.6.1 Detailed Description

Includes all other headers. Useful for rapid import.

Author

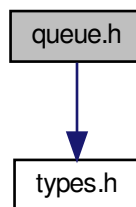
Pietro Firpo (pietro.firpo@pm.me)

5.7 queue.h File Reference

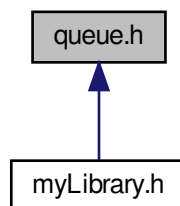
Functions for working with [Queue](#) type.

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

Include dependency graph for queue.h:



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



Functions

- [Queue newQueue](#) (const [spec_t](#) spec)
Allocate a new [Queue](#) of specified type.
- void [enqueue](#) ([Queue](#) queue,...)

- Enqueue an item into a [Queue](#).*
- void [dequeue](#) ([Queue](#) queue, void *dest)
- Dequeue an item from a [Queue](#).*
- void [printQueue](#) (const [spec_t](#) spec, const [Queue](#) queue)
- Print contents from a [Queue](#).*
- unsigned int [getQueueLength](#) (const [Queue](#) queue)
- Get the size of a [Queue](#).*
- void [deleteHeadFromQueue](#) ([Queue](#) queue)
- Delete current [Queue](#) head.*
- void [getHeadDataFromQueue](#) (const [Queue](#) queue, void *dest)
- Get the item in the head of a [Queue](#) without dequeuing it.*
- void [deleteQueue](#) ([Queue](#) queue)
- Delete a [Queue](#).*
- [byte](#) [isInQueue](#) ([Queue](#) queue,...)
- Detect if an item is inside a [Queue](#).*
- [Queue](#) [chooseNewQueueFromArray](#) (const [spec_t](#) spec, const void *arr, unsigned int size)
- Create a [Queue](#) from an array.*
- void [enqueueFromPtr](#) ([Queue](#) queue, const void *element)
- Enqueue an item into a [Queue](#).*
- [byte](#) [isQueueEmpty](#) ([Stack](#) stack)
- Check if [Queue](#) is empty.*
- [Queue](#) [newQueueFromCharArray](#) (const char arr[], unsigned int size)
- Create a [Queue](#) from an array of chars.*
- [Queue](#) [newQueueFromIntArray](#) (const int arr[], unsigned int size)
- Create a [Queue](#) from an array of integers.*
- [Queue](#) [newQueueFromArray](#) (const float arr[], unsigned int size)
- Create a [Queue](#) from an array of floats.*
- [Queue](#) [newQueueFromDoubleArray](#) (const double arr[], unsigned int size)
- Create a [Queue](#) from an array of doubles.*
- [Queue](#) [newQueueFromPtrArray](#) (const void *arr, unsigned int size)
- Create a [Queue](#) from an array of pointers.*
- [byte](#) [areQueuesEqual](#) (const [Queue](#) queue1, const [Queue](#) queue2)
- Compare two [Queue](#).*

5.7.1 Detailed Description

Functions for working with [Queue](#) type.

Author

Pietro Firpo (pietro.firpo@pm.me)

5.7.2 Function Documentation

5.7.2.1 [areQueuesEqual\(\)](#)

```
byte areQueuesEqual (
    const Queue queue1,
    const Queue queue2 )
```

Compare two [Queue](#).

Parameters

<i>queue1</i>	The first Queue you want to compare
<i>queue2</i>	The second Queue you want to compare

Returns

The result of the comparison

Return values

<i>TRUE</i>	<code>Queue1</code> and <code>Queue2</code> have equal type and equal contents
<i>FALSE</i>	<code>Queue1</code> and <code>Queue2</code> do not have equal type or equal contents

5.7.2.2 chooseNewQueueFromArray()

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

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

Parameters

<i>spec</i>	The type specifier of the array passed. Refer to spec_t for supported types
<i>arr</i>	The array you want to create a Queue from
<i>size</i>	The number of items in <code>arr</code>

Returns

A [Queue](#) containing the elements in `arr`, having the first element of `arr` as head

5.7.2.3 deleteHeadFromQueue()

```
void deleteHeadFromQueue (
    Queue queue )
```

Delete current [Queue](#) head.

Parameters

<i>queue</i>	The Queue you want to delete the head from
--------------	--

5.7.2.4 deleteQueue()

```
void deleteQueue (
    Queue queue )
```

Delete a [Queue](#).

Parameters

<i>queue</i>	The Queue you want to delete
--------------	--

5.7.2.5 dequeue()

```
void dequeue (
    Queue queue,
    void * dest )
```

Dequeue an item from a [Queue](#).

Parameters

<i>queue</i>	The Queue you want to dequeue from
<i>dest</i>	The address of the variable you want to store the dequeued item in

5.7.2.6 enqueue()

```
void enqueue (
    Queue queue,
    ... )
```

Enqueue an item into a [Queue](#).

Parameters

<i>queue</i>	The Queue you want to enqueue an item into
...	The item you want to enqueue into <i>queue</i>

Note

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

5.7.2.7 enqueueFromPtr()

```
void enqueueFromPtr (
    Queue queue,
    const void * element )
```

Enqueue an item into a [Queue](#).

Parameters

<i>queue</i>	The Queue you want to enqueue an item into
<i>element</i>	Pointer to the item you want to enqueue into <i>queue</i>

5.7.2.8 getHeadDataFromQueue()

```
void getHeadDataFromQueue (
    const Queue queue,
    void * dest )
```

Get the item in the head of a [Queue](#) without dequeuing it.

Parameters

<i>queue</i>	The Queue you want to get the item in the head from
<i>dest</i>	The address of the variable you want to store the item in

5.7.2.9 getQueueLength()

```
unsigned int getQueueLength (
    const Queue queue )
```

Get the size of a [Queue](#).

Parameters

<i>queue</i>	The Queue you want to evaluate
--------------	--

Returns

The number of elements in *queue*

5.7.2.10 isInQueue()

```
byte isInQueue (
    Queue queue,
    ... )
```

Detect if an item is inside a [Queue](#).

Parameters

<i>queue</i>	The Queue you want search in
...	The element you want to search

Note

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

Even though specifying more than one item for single call 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 <i>queue</i>
<i>FALSE</i>	Given element is not contained in <i>queue</i>

5.7.2.11 isQueueEmpty()

```
byte isQueueEmpty (
    Stack stack )
```

Check if [Queue](#) is empty.

Parameters

<i>stack</i>	The Queue to be checked
--------------	---

Return values

<i>TRUE</i>	queue is empty
<i>FALSE</i>	queue is not empty

5.7.2.12 newQueue()

```
Queue newQueue (
    const spec_t spec )
```

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

Parameters

<i>spec</i>	Type specifier of the Queue you want to create. Refer to spec_t for supported types
-------------	---

Returns

An empty [Queue](#)

5.7.2.13 `newQueueFromCharArray()`

```
Queue newQueueFromCharArray (  
    const char arr[],  
    unsigned int size )
```

Create a [Queue](#) from an array of chars.

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

5.7.2.14 `newQueueFromDoubleArray()`

```
Queue newQueueFromDoubleArray (  
    const double arr[],  
    unsigned int size )
```

Create a [Queue](#) from an array of doubles.

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

5.7.2.15 `newQueueFromFloatArray()`

```
Queue newQueueFromFloatArray (  
    const float arr[],  
    unsigned int size )
```

Create a [Queue](#) from an array of floats.

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

5.7.2.16 newQueueFromIntArray()

```
Queue newQueueFromIntArray (
    const int arr[],
    unsigned int size )
```

Create a [Queue](#) from an array of integers.

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

5.7.2.17 newQueueFromPtrArray()

```
Queue newQueueFromPtrArray (
    const void * arr,
    unsigned int size )
```

Create a [Queue](#) from an array of pointers.

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

5.7.2.18 printQueue()

```
void printQueue (
    const spec_t spec,
    const Queue queue )
```

Print contents from a [Queue](#).

Parameters

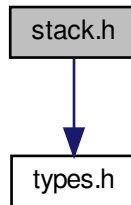
<i>spec</i>	The type and format specifier you want to use to print the single element of the Queue . Use the <code>printf()</code> conventions
<i>queue</i>	The Queue you want to print

5.8 stack.h File Reference

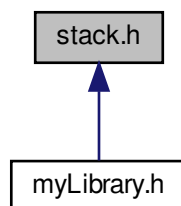
Functions for working with [Stack](#) type.

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

Include dependency graph for stack.h:



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



Functions

- `Stack newStack` (const `spec_t` spec)
Allocate a new `Stack` of specified type.
- void `push` (`Stack` stack,...)
Push an item into a `Stack`.
- void `printStack` (const `spec_t` spec, const `Stack` stack)
Print contents from a `Stack`.
- void `pop` (`Stack` stack, void *dest)
Pop an item from a `Stack`.
- void `deleteHeadFromStack` (`Stack` stack)
Delete current `Stack` head.
- `byte isStackEmpty` (`Stack` stack)
Check if `Stack` is empty.
- void `deleteStack` (`Stack` stack)
Delete a `Stack`.
- void `getHeadDataFromStack` (`Stack` stack, void *dest)
Get the item at the head of a `Stack` without popping it.

- [byte](#) [isInStack](#) ([Stack](#) stack,...)
Detect if an item is inside a [Stack](#).
- [Stack](#) [chooseNewStackFromArray](#) (const [spec_t](#) spec, const void *arr, unsigned int size)
Create a [Stack](#) from an array.
- void [pushFromPtr](#) ([Stack](#) stack, const void *element)
Push an item into a [Stack](#).
- unsigned int [getStackLength](#) (const [Stack](#) stack)
Get the size of a [Stack](#).
- [Stack](#) [newStackFromCharArray](#) (const char arr[], unsigned int size)
Create a [Stack](#) from an array of chars.
- [Stack](#) [newStackFromIntArray](#) (const int arr[], unsigned int size)
Create a [Stack](#) from an array of integers.
- [Stack](#) [newStackFromFloatArray](#) (const float arr[], unsigned int size)
Create a [Stack](#) from an array of floats.
- [Stack](#) [newStackFromDoubleArray](#) (const double arr[], unsigned int size)
Create a [Stack](#) from an array of doubles.
- [Stack](#) [newStackFromPtrArray](#) (const void *arr, unsigned int size)
Create a [Stack](#) from an array of pointers.
- [byte](#) [areStacksEqual](#) (const [Stack](#) stack1, const [Stack](#) stack2)
Compare two [Stack](#).

5.8.1 Detailed Description

Functions for working with [Stack](#) type.

Author

Pietro Firpo (pietro.firpo@pm.me)

5.8.2 Function Documentation

5.8.2.1 [areStacksEqual\(\)](#)

```
byte areStacksEqual (  
    const Stack stack1,  
    const Stack stack2 )
```

Compare two [Stack](#).

Parameters

stack1	The first Stack you want to compare
stack2	The second Stack you want to compare

Returns

The result of the comparison

Return values

<i>TRUE</i>	<code>stack1</code> and <code>stack2</code> have equal type and equal contents
<i>FALSE</i>	<code>stack1</code> and <code>stack2</code> do not have equal type or equal contents

5.8.2.2 chooseNewStackFromArray()

```
Stack chooseNewStackFromArray (
    const spec_t spec,
    const void * arr,
    unsigned int size )
```

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

Parameters

<i>spec</i>	The type specifier of the array passed. Refer to spec_t for supported types
<i>arr</i>	The array you want to create the Stack from
<i>size</i>	The number of items in <code>arr</code>

Returns

A [Stack](#) containing the elements in `arr`, having the last element of `arr` as head

5.8.2.3 deleteHeadFromStack()

```
void deleteHeadFromStack (
    Stack stack )
```

Delete current [Stack](#) head.

Parameters

<i>stack</i>	The Stack you want to delete the head from
--------------	--

5.8.2.4 deleteStack()

```
void deleteStack (
    Stack stack )
```

Delete a [Stack](#).

Parameters

<i>stack</i>	The Stack you want to delete
--------------	--

5.8.2.5 getHeadDataFromStack()

```
void getHeadDataFromStack (
    Stack stack,
    void * dest )
```

Get the item at the head of a [Stack](#) without popping it.

Parameters

<i>stack</i>	The Stack you want to get the item
<i>dest</i>	The address of the variable you want to store the item in

5.8.2.6 getStackLength()

```
unsigned int getStackLength (
    const Stack stack )
```

Get the size of a [Stack](#).

Parameters

<i>stack</i>	The Stack you want to evaluate
--------------	--

Returns

The number of elements in `stack`

5.8.2.7 isInStack()

```
byte isInStack (
    Stack stack,
    ... )
```

Detect if an item is inside a [Stack](#).

Parameters

<i>stack</i>	The Stack you want search in
<i>...</i>	The element you want to search

Note

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

Even though specifying more than one item for single call 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 <i>stack</i>
<i>FALSE</i>	Given element is not contained in <i>stack</i>

5.8.2.8 isStackEmpty()

```
byte isStackEmpty (
    Stack stack )
```

Check if [Stack](#) is empty.

Parameters

<i>stack</i>	The Stack to be checked
--------------	---

Return values

<i>TRUE</i>	<i>stack</i> is empty
<i>FALSE</i>	<i>stack</i> is not empty

5.8.2.9 newStack()

```
Stack newStack (
    const spec\_t spec )
```

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

Parameters

<i>spec</i>	Type specifier of the Stack you want to create. Refer to spec_t for supported types
-------------	---

Returns

An empty [Stack](#)

5.8.2.10 newStackFromCharArray()

```
Stack newStackFromCharArray (
    const char arr[],
    unsigned int size )
```

Create a [Stack](#) from an array of chars.

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

5.8.2.11 newStackFromDoubleArray()

```
Stack newStackFromDoubleArray (
    const double arr[],
    unsigned int size )
```

Create a [Stack](#) from an array of doubles.

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

5.8.2.12 newStackFromFloatArray()

```
Stack newStackFromFloatArray (
    const float arr[],
    unsigned int size )
```

Create a [Stack](#) from an array of floats.

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

5.8.2.13 newStackFromIntArray()

```
Stack newStackFromIntArray (
    const int arr[],
    unsigned int size )
```

Create a [Stack](#) from an array of integers.

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

5.8.2.14 newStackFromPtrArray()

```
Stack newStackFromPtrArray (
    const void * arr,
    unsigned int size )
```

Create a [Stack](#) from an array of pointers.

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

5.8.2.15 pop()

```
void pop (
    Stack stack,
    void * dest )
```

Pop an item from a [Stack](#).

Parameters

<i>stack</i>	The Stack you want to pop an item from
<i>dest</i>	The address of the variable you want to store the popped item in

5.8.2.16 printStack()

```
void printStack (
    const spec_t spec,
    const Stack stack )
```

Print contents from a [Stack](#).

Parameters

<i>spec</i>	The type and format specifier you want to use to print the single element of the Stack . Use the <code>printf()</code> conventions
<i>stack</i>	The Stack you want to print

5.8.2.17 push()

```
void push (
    Stack stack,
    ... )
```

Push an item into a [Stack](#).

Parameters

<i>stack</i>	The Stack you want to push into
<i>...</i>	The item you want to push into <i>stack</i>

Note

Even though pushing more than one item for single call does not throw a compiler nor runtime error, only pushing one item is supported. Other items are ignored and are not pushed into *stack*. If you don't specify any item to be pushed, still no errors occur but the content of your [Stack](#) can be messed up

5.8.2.18 pushFromPtr()

```
void pushFromPtr (
    Stack stack,
    const void * element )
```

Push an item into a [Stack](#).

Parameters

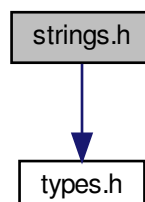
<i>stack</i>	The Stack you want to push an item into
<i>element</i>	Pointer to the item you want to push into <i>stack</i>

5.9 strings.h File Reference

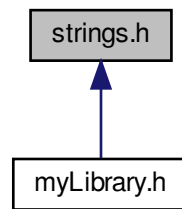
Common tasks with strings.

```
#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 a tring with different last character.
- `string copyOf (const string src)`
Get a copy of the given `string`.

5.9.1 Detailed Description

Common tasks with strings.

Author

Pietro Firpo (pietro.firpo@pm.me)

5.9.2 Function Documentation

5.9.2.1 changeLastCharacter()

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

Get a tring 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 the return code of the function

Return values

<i>NULL</i>	Errors occurred during the execution of the function
-------------	--

5.9.2.2 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](#)

5.9.2.3 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

The return code of the function

Return values

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

5.9.2.4 getString()

```
string getString ( )
```

Reads from terminal a string of arbitrary length.

Returns

A char pointer to the first character of the string or the return code of the function

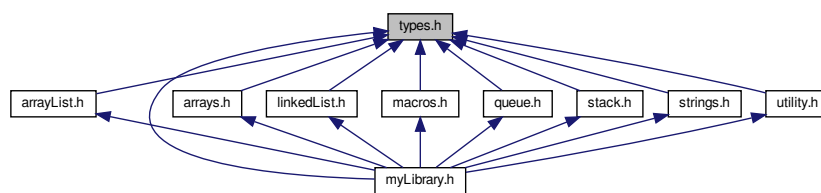
Return values

<i>NULL</i>	Errors occurred during the execution of the function
-------------	--

5.10 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
- struct [Stack](#)
Stack type
- struct [Queue](#)
Queue 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

5.10.1 Detailed Description

Collection of useful types.

Author

Pietro Firpo (pietro.firpo@pm.me)

5.10.2 Typedef Documentation

5.10.2.1 byte

```
typedef char byte
```

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

5.10.2.2 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

5.10.2.3 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

5.10.2.4 string

```
typedef char* string
```

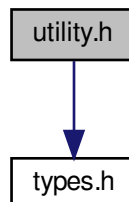
Alias for char *, used when an array of char is actually used as a string.

5.11 utility.h File Reference

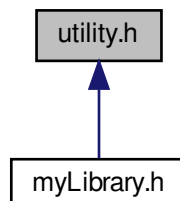
Common tasks such as comparing variables, allocate memory.

```
#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.

5.11.1 Detailed Description

Common tasks such as comparing variables, allocate memory.

Author

Pietro Firpo (pietro.firpo@pm.me)

5.11.2 Function Documentation

5.11.2.1 byteCmp()

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

Compare two bytes.

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

5.11.2.2 charCmp()

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

Compare two chars.

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

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

5.11.2.4 doubleCmp()

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

Compare two doubles.

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

5.11.2.5 floatCmp()

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

Compare two floats.

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

5.11.2.6 intCmp()

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

Compare two ints.

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

5.11.2.7 ptrCmp()

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

Compare two pointers.

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

5.11.2.8 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

5.11.2.9 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

- append
 - macros.h, [48](#)
- appendToAL
 - arrayList.h, [17](#)
- appendToLL
 - linkedList.h, [37](#)
- appendToLLFromPtr
 - linkedList.h, [37](#)
- areALEqual
 - arrayList.h, [17](#)
- areEqual
 - macros.h, [48](#)
- areLLEqual
 - linkedList.h, [37](#)
- areQueuesEqual
 - queue.h, [61](#)
- areStacksEqual
 - stack.h, [69](#)
- ArrayList, [7](#)
 - body, [7](#)
 - size, [7](#)
 - type, [8](#)
- arrayList.h, [15](#)
 - appendToAL, [17](#)
 - areALEqual, [17](#)
 - bubbleSortAL, [18](#)
 - chooseNewALFromArray, [18](#)
 - deleteAL, [18](#)
 - getALLength, [19](#)
 - getFromAL, [19](#)
 - insertToAL, [19](#)
 - isALEmpty, [21](#)
 - isInAL, [21](#)
 - linearSearchAL, [22](#)
 - mergeAL, [22](#)
 - newAL, [23](#)
 - newALFromAL, [23](#)
 - newALFromByteArray, [23](#)
 - newALFromCharArray, [23](#)
 - newALFromDoubleArray, [24](#)
 - newALFromFloatArray, [24](#)
 - newALFromIntArray, [24](#)
 - newALFromPtrArray, [24](#)
 - printAL, [24](#)
 - quickSortAL, [25](#)
 - removeFromAL, [25](#)
 - reverseAL, [25](#)
 - setALItem, [26](#)
 - sliceAL, [26](#)
- arrays.h, [26](#)
 - charBubbleSort, [28](#)
 - charLinearSearch, [28](#)
 - charQuickSort, [29](#)
 - chooseBubbleSortArr, [29](#)
 - chooseLinearSearch, [29](#)
 - chooseQuickSortArr, [30](#)
 - doubleBubbleSort, [30](#)
 - doubleLinearSearch, [30](#)
 - doubleQuickSort, [31](#)
 - floatBubbleSort, [31](#)
 - floatLinearSearch, [31](#)
 - floatQuickSort, [31](#)
 - intBubbleSort, [31](#)
 - intLinearSearch, [32](#)
 - intQuickSort, [32](#)
 - printMatrix, [32](#)
 - ptrBubbleSort, [33](#)
 - ptrLinearSearch, [33](#)
 - ptrQuickSort, [33](#)
- body
 - ArrayList, [7](#)
- bubbleSortAL
 - arrayList.h, [18](#)
- bubbleSortArr
 - macros.h, [48](#)
- byte
 - types.h, [79](#)
- byteCmp
 - utility.h, [81](#)
- changeLastCharacter
 - strings.h, [76](#)
- charBubbleSort
 - arrays.h, [28](#)
- charCmp
 - utility.h, [81](#)
- charLinearSearch
 - arrays.h, [28](#)
- charQuickSort
 - arrays.h, [29](#)
- chooseBubbleSortArr
 - arrays.h, [29](#)
- chooseCmp
 - utility.h, [81](#)
- chooseLinearSearch
 - arrays.h, [29](#)
- chooseNewALFromArray
 - arrayList.h, [18](#)

- chooseNewLLFromArray
 - linkedList.h, 38
- chooseNewQueueFromArray
 - queue.h, 62
- chooseNewStackFromArray
 - stack.h, 70
- chooseQuickSortArr
 - arrays.h, 30
- cmpVal
 - macros.h, 49
- constants.h, 33
 - EQUAL, 34
 - FALSE, 34
 - GREATER, 34
 - KEY_NOT_FOUND, 34
 - SMALLER, 35
 - TRUE, 35
- copyOf
 - strings.h, 77
- data
 - node, 10
- delete
 - macros.h, 50
- deleteAL
 - arrayList.h, 18
- deleteHead
 - macros.h, 50
- deleteHeadFromQueue
 - queue.h, 62
- deleteHeadFromStack
 - stack.h, 70
- deleteLL
 - linkedList.h, 38
- deleteQueue
 - queue.h, 63
- deleteStack
 - stack.h, 70
- dequeue
 - queue.h, 63
- doubleBubbleSort
 - arrays.h, 30
- doubleCmp
 - utility.h, 82
- doubleLinearSearch
 - arrays.h, 30
- doubleQuickSort
 - arrays.h, 31
- endsWith
 - strings.h, 77
- enqueue
 - queue.h, 63
- enqueueFromPtr
 - queue.h, 64
- EQUAL
 - constants.h, 34
- FALSE
 - constants.h, 34
- floatBubbleSort
 - arrays.h, 31
- floatCmp
 - utility.h, 82
- floatLinearSearch
 - arrays.h, 31
- floatQuickSort
 - arrays.h, 31
- getALLength
 - arrayList.h, 19
- getFromAL
 - arrayList.h, 19
- getFromLL
 - linkedList.h, 39
- getHeadData
 - macros.h, 50
- getHeadDataFromQueue
 - queue.h, 64
- getHeadDataFromStack
 - stack.h, 71
- getItem
 - macros.h, 51
- getLength
 - macros.h, 51
- getLLLength
 - linkedList.h, 39
- getQueueLength
 - queue.h, 64
- getStackLength
 - stack.h, 71
- getString
 - strings.h, 78
- GREATER
 - constants.h, 34
- head
 - LinkedList, 9
 - Queue, 12
 - Stack, 13
- insert
 - macros.h, 52
- insertToAL
 - arrayList.h, 19
- insertToLL
 - linkedList.h, 39
- intBubbleSort
 - arrays.h, 31
- intCmp
 - utility.h, 82
- intLinearSearch
 - arrays.h, 32
- intQuickSort
 - arrays.h, 32
- isALEmpty
 - arrayList.h, 21
- isEmpty

- macros.h, 52
- isIn
 - macros.h, 52
- isInAL
 - arrayList.h, 21
- isInLL
 - linkedList.h, 40
- isInQueue
 - queue.h, 64
- isInStack
 - stack.h, 71
- isLLEmpty
 - linkedList.h, 40
- isQueueEmpty
 - queue.h, 65
- isStackEmpty
 - stack.h, 72
- KEY_NOT_FOUND
 - constants.h, 34
- linearSearch
 - macros.h, 53
- linearSearchAL
 - arrayList.h, 22
- linearSearchLL
 - linkedList.h, 41
- linearSearchLLPtr
 - linkedList.h, 41
- linked
 - node, 10
- LinkedList, 8
 - head, 9
 - size, 9
 - tail, 9
 - type, 9
- linkedList.h, 35
 - appendToLL, 37
 - appendToLLFromPtr, 37
 - areLLEqual, 37
 - chooseNewLLFromArray, 38
 - deleteLL, 38
 - getFromLL, 39
 - getLLLength, 39
 - insertToLL, 39
 - isInLL, 40
 - isLLEmpty, 40
 - linearSearchLL, 41
 - linearSearchLLPtr, 41
 - mergeLL, 42
 - newLL, 42
 - newLLFromCharArray, 43
 - newLLFromDoubleArray, 43
 - newLLFromFloatArray, 43
 - newLLFromIntArray, 43
 - newLLFromLL, 43
 - newLLFromPtrArray, 44
 - printLL, 44
 - removeFromLL, 44
 - setLLItem, 45
 - sliceLL, 45
- macros.h, 45
 - append, 48
 - areEqual, 48
 - bubbleSortArr, 48
 - cmpVal, 49
 - delete, 50
 - deleteHead, 50
 - getHeadData, 50
 - getItem, 51
 - getLength, 51
 - insert, 52
 - isEmpty, 52
 - isIn, 52
 - linearSearch, 53
 - merge, 54
 - newALFromArray, 54
 - newLLFromArray, 55
 - newQueueFromArray, 55
 - newStackFromArray, 56
 - print, 57
 - quickSortArr, 57
 - removeItem, 58
 - set, 58
 - slice, 59
- merge
 - macros.h, 54
- mergeAL
 - arrayList.h, 22
- mergeLL
 - linkedList.h, 42
- myLibrary.h, 59
- newAL
 - arrayList.h, 23
- newALFromAL
 - arrayList.h, 23
- newALFromArray
 - macros.h, 54
- newALFromByteArray
 - arrayList.h, 23
- newALFromCharArray
 - arrayList.h, 23
- newALFromDoubleArray
 - arrayList.h, 24
- newALFromFloatArray
 - arrayList.h, 24
- newALFromIntArray
 - arrayList.h, 24
- newALFromPtrArray
 - arrayList.h, 24
- newLL
 - linkedList.h, 42
- newLLFromArray
 - macros.h, 55
- newLLFromCharArray
 - linkedList.h, 43

- newLLFromDoubleArray
 - linkedList.h, [43](#)
- newLLFromFloatArray
 - linkedList.h, [43](#)
- newLLFromIntArray
 - linkedList.h, [43](#)
- newLLFromLL
 - linkedList.h, [43](#)
- newLLFromPtrArray
 - linkedList.h, [44](#)
- newQueue
 - queue.h, [65](#)
- newQueueFromArray
 - macros.h, [55](#)
- newQueueFromCharArray
 - queue.h, [66](#)
- newQueueFromDoubleArray
 - queue.h, [66](#)
- newQueueFromFloatArray
 - queue.h, [66](#)
- newQueueFromIntArray
 - queue.h, [66](#)
- newQueueFromPtrArray
 - queue.h, [67](#)
- newStack
 - stack.h, [72](#)
- newStackFromArray
 - macros.h, [56](#)
- newStackFromCharArray
 - stack.h, [73](#)
- newStackFromDoubleArray
 - stack.h, [73](#)
- newStackFromFloatArray
 - stack.h, [73](#)
- newStackFromIntArray
 - stack.h, [73](#)
- newStackFromPtrArray
 - stack.h, [73](#)
- Node
 - types.h, [79](#)
- node, [10](#)
 - data, [10](#)
 - linked, [10](#)
- pop
 - stack.h, [74](#)
- print
 - macros.h, [57](#)
- printAL
 - arrayList.h, [24](#)
- printLL
 - linkedList.h, [44](#)
- printMatrix
 - arrays.h, [32](#)
- printQueue
 - queue.h, [67](#)
- printStack
 - stack.h, [74](#)
- ptrBubbleSort
 - arrays.h, [33](#)
- ptrCmp
 - utility.h, [83](#)
- ptrLinearSearch
 - arrays.h, [33](#)
- ptrQuickSort
 - arrays.h, [33](#)
- push
 - stack.h, [74](#)
- pushFromPtr
 - stack.h, [75](#)
- Queue, [11](#)
 - head, [12](#)
 - size, [12](#)
 - tail, [12](#)
 - type, [12](#)
- queue.h, [60](#)
 - areQueuesEqual, [61](#)
 - chooseNewQueueFromArray, [62](#)
 - deleteHeadFromQueue, [62](#)
 - deleteQueue, [63](#)
 - dequeue, [63](#)
 - enqueue, [63](#)
 - enqueueFromPtr, [64](#)
 - getHeadDataFromQueue, [64](#)
 - getQueueLength, [64](#)
 - isInQueue, [64](#)
 - isEmpty, [65](#)
 - newQueue, [65](#)
 - newQueueFromCharArray, [66](#)
 - newQueueFromDoubleArray, [66](#)
 - newQueueFromFloatArray, [66](#)
 - newQueueFromIntArray, [66](#)
 - newQueueFromPtrArray, [67](#)
 - printQueue, [67](#)
- quickSortAL
 - arrayList.h, [25](#)
- quickSortArr
 - macros.h, [57](#)
- removeFromAL
 - arrayList.h, [25](#)
- removeFromLL
 - linkedList.h, [44](#)
- removeItem
 - macros.h, [58](#)
- reverseAL
 - arrayList.h, [25](#)
- saferMalloc
 - utility.h, [83](#)
- saferRealloc
 - utility.h, [83](#)
- set
 - macros.h, [58](#)
- setALItem
 - arrayList.h, [26](#)
- setLLItem

- linkedList.h, [45](#)
- size
 - ArrayList, [7](#)
 - LinkedList, [9](#)
 - Queue, [12](#)
- slice
 - macros.h, [59](#)
- sliceAL
 - arrayList.h, [26](#)
- sliceLL
 - linkedList.h, [45](#)
- SMALLER
 - constants.h, [35](#)
- spec_t
 - types.h, [79](#)
- Stack, [13](#)
 - head, [13](#)
 - type, [13](#)
- stack.h, [67](#)
 - areStacksEqual, [69](#)
 - chooseNewStackFromArray, [70](#)
 - deleteHeadFromStack, [70](#)
 - deleteStack, [70](#)
 - getHeadDataFromStack, [71](#)
 - getStackLength, [71](#)
 - isInStack, [71](#)
 - isStackEmpty, [72](#)
 - newStack, [72](#)
 - newStackFromCharArray, [73](#)
 - newStackFromDoubleArray, [73](#)
 - newStackFromFloatArray, [73](#)
 - newStackFromIntArray, [73](#)
 - newStackFromPtrArray, [73](#)
 - pop, [74](#)
 - printStack, [74](#)
 - push, [74](#)
 - pushFromPtr, [75](#)
- string
 - types.h, [80](#)
- strings.h, [75](#)
 - changeLastCharacter, [76](#)
 - copyOf, [77](#)
 - endsWith, [77](#)
 - getString, [78](#)
- tail
 - LinkedList, [9](#)
 - Queue, [12](#)
- TRUE
 - constants.h, [35](#)
- type
 - ArrayList, [8](#)
 - LinkedList, [9](#)
 - Queue, [12](#)
 - Stack, [13](#)
- types.h, [78](#)
 - byte, [79](#)
 - Node, [79](#)
 - spec_t, [79](#)
- string, [80](#)
- utility.h, [80](#)
 - byteCmp, [81](#)
 - charCmp, [81](#)
 - chooseCmp, [81](#)
 - doubleCmp, [82](#)
 - floatCmp, [82](#)
 - intCmp, [82](#)
 - ptrCmp, [83](#)
 - saferMalloc, [83](#)
 - saferRealloc, [83](#)