

My library

Generated by Doxygen 1.9.2

1 Hi, welcome to myLibrary!	1
1.1 Table of contents	1
1.2 Introduction and examples	1
1.2.1 ArrayLists	2
1.2.2 LinkedLists	2
1.2.3 Stacks	3
1.2.4 Queues	4
1.2.5 Array algorithms	5
1.2.6 Strings	5
1.2.7 Miscellaneous	5
1.3 How to import	6
1.3.1 On Linux	6
1.3.2 On Visual Studio for Windows	6
1.4 How to compile from source	7
2 Data Structure Index	9
2.1 Data Structures	9
3 File Index	11
3.1 File List	11
4 Data Structure Documentation	13
4.1 ArrayList Struct Reference	13
4.1.1 Detailed Description	13
4.1.2 Field Documentation	13
4.1.2.1 body	13
4.1.2.2 size	14
4.1.2.3 type	14
4.2 LinkedList Struct Reference	14
4.2.1 Detailed Description	15
4.2.2 Field Documentation	15
4.2.2.1 head	15
4.2.2.2 size	15
4.2.2.3 tail	15
4.2.2.4 type	15
4.3 node Struct Reference	16
4.3.1 Detailed Description	16
4.3.2 Field Documentation	16
4.3.2.1 data	16
4.3.2.2 linked	17
4.4 Queue Struct Reference	17
4.4.1 Detailed Description	17
4.4.2 Field Documentation	18

4.4.2.1 head	18
4.4.2.2 size	18
4.4.2.3 tail	18
4.4.2.4 type	18
4.5 Stack Struct Reference	19
4.5.1 Detailed Description	19
4.5.2 Field Documentation	19
4.5.2.1 head	19
4.5.2.2 type	20
5 File Documentation	21
5.1 arrayList.h File Reference	21
5.1.1 Detailed Description	23
5.1.2 Function Documentation	23
5.1.2.1 appendToAL()	23
5.1.2.2 areALEqual()	23
5.1.2.3 bubbleSortAL()	24
5.1.2.4 chooseNewALFromArray()	24
5.1.2.5 deleteAL()	25
5.1.2.6 getALLength()	25
5.1.2.7 getFromAL()	26
5.1.2.8 insertToAL()	26
5.1.2.9 isEmpty()	26
5.1.2.10 islnAL()	28
5.1.2.11 linearSearchAL()	28
5.1.2.12 mergeAL()	29
5.1.2.13 newAL()	29
5.1.2.14 newALFromAL()	30
5.1.2.15 newALFromByteArray()	30
5.1.2.16 newALFromCharArray()	30
5.1.2.17 newALFromDoubleArray()	31
5.1.2.18 newALFromFloatArray()	31
5.1.2.19 newALFromIntArray()	31
5.1.2.20 newALFromPtrArray()	31
5.1.2.21 printAL()	31
5.1.2.22 quickSortAL()	32
5.1.2.23 removeFromAL()	32
5.1.2.24 reverseAL()	33
5.1.2.25 setALItem()	33
5.1.2.26 sliceAL()	33
5.2 arrayList.h	34
5.3 arrays.h File Reference	34

5.3.1 Detailed Description	36
5.3.2 Function Documentation	36
5.3.2.1 charBubbleSortArr()	36
5.3.2.2 charLinearSearchArr()	37
5.3.2.3 charQuickSortArr()	37
5.3.2.4 chooseBubbleSortArr()	37
5.3.2.5 chooseLinearSearchArr()	37
5.3.2.6 chooseQuickSortArr()	38
5.3.2.7 doubleBubbleSortArr()	39
5.3.2.8 doubleLinearSearchArr()	39
5.3.2.9 doubleQuickSortArr()	39
5.3.2.10 floatBubbleSortArr()	39
5.3.2.11 floatLinearSearchArr()	39
5.3.2.12 floatQuickSortArr()	40
5.3.2.13 intBubbleSortArr()	40
5.3.2.14 intLinearSearchArr()	40
5.3.2.15 intQuickSortArr()	40
5.3.2.16 printMatrix()	40
5.3.2.17 ptrBubbleSortArr()	41
5.3.2.18 ptrLinearSearchArr()	41
5.3.2.19 ptrQuickSortArr()	42
5.4 arrays.h	42
5.5 constants.h File Reference	43
5.5.1 Detailed Description	44
5.5.2 Macro Definition Documentation	44
5.5.2.1 EQUAL	44
5.5.2.2 FALSE	44
5.5.2.3 GREATER	44
5.5.2.4 KEY_NOT_FOUND	45
5.5.2.5 SMALLER	45
5.5.2.6 TRUE	45
5.6 constants.h	45
5.7 linkedList.h File Reference	46
5.7.1 Detailed Description	47
5.7.2 Function Documentation	47
5.7.2.1 appendToLL()	48
5.7.2.2 appendToLLFromPtr()	48
5.7.2.3 areLLEqual()	48
5.7.2.4 chooseNewLLFromArray()	49
5.7.2.5 deleteLL()	49
5.7.2.6 getFromLL()	49
5.7.2.7 getLLLlength()	50

5.7.2.8 insertToLL()	50
5.7.2.9 isInLL()	51
5.7.2.10 isLLEmpty()	51
5.7.2.11 linearSearchLL()	52
5.7.2.12 linearSearchLLPtr()	52
5.7.2.13 mergeLL()	53
5.7.2.14 newLL()	53
5.7.2.15 newLLFromCharArray()	53
5.7.2.16 newLLFromDoubleArray()	54
5.7.2.17 newLLFromFloatArray()	54
5.7.2.18 newLLFromIntArray()	54
5.7.2.19 newLLFromLL()	54
5.7.2.20 newLLFromPtrArray()	55
5.7.2.21 printLL()	55
5.7.2.22 removeFromLL()	55
5.7.2.23 setLLItem()	55
5.7.2.24 sliceLL()	56
5.8 linkedList.h	56
5.9 macros.h File Reference	57
5.9.1 Detailed Description	59
5.9.2 Macro Definition Documentation	59
5.9.2.1 append	59
5.9.2.2 areEqual	60
5.9.2.3 bubbleSortArr	60
5.9.2.4 cmpVal	61
5.9.2.5 delete	61
5.9.2.6 deleteHead	62
5.9.2.7 getItem	62
5.9.2.8 getLength	62
5.9.2.9 insert	63
5.9.2.10 isEmpty	63
5.9.2.11 isIn	64
5.9.2.12 linearSearch	64
5.9.2.13 merge	66
5.9.2.14 newALFromArray	66
5.9.2.15 newLLFromArray	67
5.9.2.16 newQueueFromArray	67
5.9.2.17 newStackFromArray [1/2]	68
5.9.2.18 newStackFromArray [2/2]	69
5.9.2.19 peek	69
5.9.2.20 print	70
5.9.2.21 quickSortArr	70

5.9.2.22 removeItem	71
5.9.2.23 set	71
5.9.2.24 slice	71
5.10 macros.h	72
5.11 mainPage.md File Reference	74
5.12 myLibrary.h File Reference	74
5.12.1 Detailed Description	74
5.13 myLibrary.h	74
5.14 queue.h File Reference	75
5.14.1 Detailed Description	76
5.14.2 Function Documentation	76
5.14.2.1 areQueuesEqual()	76
5.14.2.2 chooseNewQueueFromArray()	77
5.14.2.3 deleteHeadFromQueue()	77
5.14.2.4 deleteQueue()	77
5.14.2.5 dequeue()	78
5.14.2.6 enqueue()	78
5.14.2.7 enqueueFromPtr()	78
5.14.2.8 getQueueLength()	80
5.14.2.9 isInQueue()	80
5.14.2.10 isEmptyQueue()	81
5.14.2.11 newQueue()	81
5.14.2.12 newQueueFromCharArray()	81
5.14.2.13 newQueueFromDoubleArray()	82
5.14.2.14 newQueueFromFloatArray()	82
5.14.2.15 newQueueFromIntArray()	82
5.14.2.16 newQueueFromPtrArray()	82
5.14.2.17 peekQueue()	82
5.14.2.18 printQueue()	83
5.15 queue.h	83
5.16 stack.h File Reference	84
5.16.1 Detailed Description	85
5.16.2 Function Documentation	85
5.16.2.1 areStacksEqual()	85
5.16.2.2 chooseNewStackFromArray()	86
5.16.2.3 deleteHeadFromStack()	86
5.16.2.4 deleteStack()	87
5.16.2.5 getStackLength()	87
5.16.2.6 isInStack()	87
5.16.2.7 isEmptyStack()	88
5.16.2.8 newStack()	88
5.16.2.9 newStackFromCharArray()	89

5.16.2.10 newStackFromDoubleArray()	89
5.16.2.11 newStackFromFloatArray()	89
5.16.2.12 newStackFromIntArray()	89
5.16.2.13 newStackFromPtrArray()	89
5.16.2.14 peekStack()	89
5.16.2.15 pop()	90
5.16.2.16 printStack()	90
5.16.2.17 push()	90
5.16.2.18 pushFromPtr()	91
5.17 stack.h	91
5.18 strings.h File Reference	92
5.18.1 Detailed Description	93
5.18.2 Function Documentation	93
5.18.2.1 changeLastCharacter()	93
5.18.2.2 copyOf()	93
5.18.2.3 endsWith()	94
5.18.2.4 getString()	94
5.19 strings.h	95
5.20 types.h File Reference	95
5.20.1 Detailed Description	96
5.20.2 Typedef Documentation	96
5.20.2.1 byte	96
5.20.2.2 Node	96
5.20.2.3 spec_t	96
5.20.2.4 string	97
5.21 types.h	97
5.22 utility.h File Reference	97
5.22.1 Detailed Description	99
5.22.2 Function Documentation	99
5.22.2.1 byteCmp()	99
5.22.2.2 charCmp()	99
5.22.2.3 chooseCmp()	99
5.22.2.4 doubleCmp()	100
5.22.2.5 floatCmp()	100
5.22.2.6 intCmp()	100
5.22.2.7 ptrCmp()	100
5.22.2.8 saferMalloc()	100
5.22.2.9 saferRealloc()	101
5.23 utility.h	101

Chapter 1

Hi, welcome to myLibrary!

This is C library with some common tasks and data structures. I know the name is not the best but I have no imagination for names.

Check [here](#) for the documentation and [here](#) to download latest build (for x86-64 only, but you can compile from source in order to support other architectures).

1.1 Table of contents

- [Introduction and examples](#)
 - [ArrayLists](#)
 - [LinkedLists](#)
 - [Stacks](#)
 - [Queues](#)
 - [Array algorithms](#)
 - [Strings](#)
 - [Miscellaneous](#)
- [How to import](#)
 - [On Linux](#)
 - [On Visual Studio for Windows](#)
- [How to compile from source](#)

1.2 Introduction and examples

This library contains some useful data structures which are not supported by default in C and some frequently used functions and algorithms.

As this library is written in C, almost every function needs you to specify as a function argument which type of data you are using it on through type and formatting specifiers. The convention used is the same used in standard C for `printf` and `scanf`.

In order to make writing code a bit more lighter the library includes also some macros that automatically detect the type of arguments passed so you don't have to explicitly use format and type specifiers. However, these macros are supported on C11 and newer compilers only and using them in some development environments can cause warnings or error reportings even though they are used correctly. See [macros.h](#) in the docs for more. In order to be as inclusive as possible, since this macros are not supported by every C compilers, in the following examples they are not used.

The approach used by this library to handle errors with pointers is that every error is fatal: for example, when a pointer passed to a function is null and it should not be null, or some needed memory could not be allocated, the program prints where the error occurred and exits.

1.2.1 ArrayLists

ArrayLists are dynamically growing and shrinking lists of data, which can be of `char`, `int`, `float`, `double` or pointer type. You can create an [ArrayList](#) from a C array or you can create a new empty [ArrayList](#). You can append items at its end, insert items, change its items, get its items, sort it (only ascending order is currently supported), print it, merge it with another [ArrayList](#) and much more. See [arrayList.h](#) in the docs for all the details.

The difference with [LinkedLists](#) is in the implementation and hence in the time needed for accessing its item. For example, an [ArrayList](#) has constant time for accessing items, while a [LinkedList](#) takes linear time. If you are interested in these topics I suggest you to search more information on the Internet, as [LinkedList](#) and [ArrayList](#) are very standard data structures and on the web you can find a lot of information.

Here are some examples of [ArrayList](#) usage:

```
#include "myLibrary.h"
int main() {
    // Create an empty ArrayList of int type
    ArrayList list1 = newAL("%i");
    // Print list1
    printAL("%i\n", list1);
    // Output:
    // Empty
    // Append two items to list1
    appendToAL(list1, 3);
    appendToAL(list1, 4);
    // Now list1 contains: 3, 4
    // Insert an item to list1 at index 1
    insertToAL(list1, 1, -1);
    // Now list1 contains: 3, -1, 4
    // Change value of item at index 1 in list1
    setALItem(list1, 1, -2);
    // Now list1 contains: 3, -2, 4
    // Remove item at index 1 from list1
    removeFromAL(list1, 1);
    // Now list1 contains: 3, 4
    int extracted;
    // Get item at index 1 from list1 and save it into extracted
    getFromAL(list1, 1, &extracted);
    // list1 still contains: 3, 4; extracted is now 4
    int myArray[] = {23, 4, 65, -5, 12};
    // Create an ArrayList of ints from the static array myArray which contains 5 elements
    ArrayList list2 = chooseNewALFromArray("%i", myArray, 5);
    // Now list2 contains: 23, 4, 65, -5, 12
    printAL("% i", list2);
    // Output:
    // 23 4 65 -5 12
    // Sort list2 using a quicksort algorithm
    quickSortAL(list2);
    // Now list2 contains: -5, 4, 12, 23, 65
    // Reverse an ArrayList
    reverseAL(list2);
    // Now list2 contains: 65, 23, 12, 4, -5
    // Get the index of 12 in list2
    int index = linearSearchAL(list2, 12);
    // index is now 2
    // Check if list1 and list2 have equal contents
    byte areEqual = areALEqual(list1, list2);
    // areListsEqual is now FALSE (See
    // [constants.h](https://catomaior.github.io/myLibrary/constants_8h.html) docs for its numeric value)
    // Merge list1 and list2
    mergeAL(list1, list2);
    // list2 still contains: 65, 23, 12, 4, -5; list1 now contains 3, 4, 65, 23, 12, 4, -5
    // Get list1 length
    unsigned int list1Length = getALLength(list1);
    // list1Length is now 7
    // Delete list1 and list2
    deleteAL(list1);
    deleteAL(list2);
    // Memory used by list1 and list2 is now freed. In order to avoid memory leaks is always good practice
    // to delete ArrayLists before they go out of their scope
    return 0;
}
```

1.2.2 LinkedLists

LinkedLists are a quite standard implementation of linked lists, dynamically growing and shrinking lists of data, which can be of `char`, `int`, `float`, `double` or pointer type. You can create a [LinkedList](#) from a C array or you can create a new empty [LinkedList](#). You can append items at its end, insert items, change its items, get its items,

print it, merge it with another [LinkedList](#) and much more. See [LinkedList.h](#) in the docs for all the details. The difference with [ArrayLists](#) is in the implementation and hence in the time needed for accessing its item. For example, a [LinkedList](#) has constant time for accessing items, while a [LinkedList](#) takes linear time. If you are interested in these topics I suggest you to search more information on the Internet, as [ArrayLists](#) and [LinkedLists](#) are very standard data structures and on the web you can find a lot of information. As for now, [LinkedLists](#) and [ArrayLists](#) have more or less the same functionalities except sorting and reversing, which are currently supported only on [ArrayLists](#).

Here are some examples of [LinkedList](#) usage:

```
#include "myLibrary.h"
int main() {
    // Create an empty LinkedList of int type
    LinkedList list1 = newLL("%i");
    // Print list1
    printLL("%i\n", list1);
    // Output:
    // Empty
    // Append two items to list1
    appendToLL(list1, 3);
    appendToLL(list1, 4);
    // Now list1 contains: 3, 4
    // Insert an item to list1 at index 1
    insertToLL(list1, 1, -1);
    // Now list1 contains: 3, -1, 4
    // Change value of item at index 1 in list1
    setLLItem(list1, 1, -2);
    // Now list1 contains: 3, -2, 4
    // Remove item at index 1 from list1
    removeFromLL(list1, 1);
    // Now list1 contains: 3, 4
    int extracted;
    // Get item at index 1 from list1 and save it into extracted
    getFromLL(list1, 1, &extracted);
    // list1 still contains: 3, 4; extracted is now 4
    int myArray[] = {23, 4, 65, -5, 12};
    // Create a LinkedList of ints from the static array myArray which contains 5 elements
    LinkedList list2 = chooseNewLLFromArray("%i", myArray, 5);
    // Now list2 contains: 23, 4, 65, -5, 12
    printLL("%i", list1);
    // Output:
    // 23 4 65 -5 12
    // Get the index of 12 in list2
    int index = linearSearchLL(list2, 12);
    // index is now 2
    // Check if list1 and list2 have equal contents
    byte areEqual = areLLEqual(list1, list2);
    // areListsEqual is now FALSE (See
    // [constants.h](https://catomaioir.github.io/myLibrary/constants_8h.html) docs for its numeric value)
    // Merge list1 and list2
    mergeLL(list1, list2);
    // list2 still contains: 65, 23, 12, 4, -5; list1 now contains 3, 4, 65, 23, 12, 4, -5
    // Get list1 length
    unsigned int list1Length = getLLLength(list1);
    // list1Length is now 7
    // Delete list1 and list2
    deleteLL(list1);
    deleteLL(list2);
    // Memory used by list1 and list2 is now freed. In order to avoid memory leaks is always good practice
    // to delete LinkedLists before they go out of their scope
    return 0;
}
```

1.2.3 Stacks

Stacks are a quite standard implementation of LIFO stacks and can contain `char`, `int`, `float`, `double` or pointer data. You can create a [Stack](#) from a C array or you can create a new empty [Stack](#). You can print its content, push items to its top, pop items from its top, peek from its top and much more. See [stack.h](#) in the docs for all the details.

Here are some examples of [Stack](#) usage:

```
#include "myLibrary.h"
int main() {
    // Create an empty Stack of int type
    Stack stack1 = newStack("%i");
    // Print stack1
    printStack("%i\n", stack1);
    // Output:
    // Empty
    // Push three items to stack1
```

```

push(stack1, 3);
push(stack1, 4);
push(stack1, -1);
// Now stack1 contains: -1, 4, 3
int extracted;
// Pop the item on top from stack1 and save it into extracted
pop(stack1, &extracted);
// Now stack1 contains: 4, 3; extracted is now -1
// Peek the item on top from stack1 and save it into extracted
peekStack(stack1, &extracted);
// stack1 still contains: 4, 3; extracted is now 4
int myArray[] = {23, 4, 65, -5, 12};
// Create a Stack of ints from the static array myArray which contains 5 elements
Stack stack2 = chooseNewStackFromArray("%i", myArray, 5);
// Now stack2 contains: 12, -5, 65, 4, 23
printStack("% i", stack1);
// Output:
// 12 -5 65 4 23
// Check if stack1 and stack2 have equal contents
byte areEqual = areStacksEqual(stack1, stack2);
// areListsEqual is now FALSE (See
// [constants.h](https://catomaior.github.io/myLibrary/constants_8h.html) docs for its numeric value)
// Delete an item from the top of stack2 without saving it
deleteHeadFromStack(stack2);
// Now stack2 contains: -5, 65, 4, 23
// Get stack2 length
unsigned int stack2Length = getStackLength(stack2);
// stack2Length is now 4
// Delete stack1 and stack2
deleteStack(stack1);
deleteStack(stack2);
// Memory used by stack1 and stack2 is now freed. In order to avoid memory leaks deleting Stacks before
// they go out of their scope is always good practice
return 0;
}

```

1.2.4 Queues

Queues are a quite standard implementation of FIFO queues and can contain char, int, float, double or pointer data. You can create a [Queue](#) from a C array or you can create a new empty [Queue](#). You can print its content, enqueue items to its end, dequeue items from its top, peek from its top and much more. See [queue.h](#) in the docs for all the details.

Here are some examples of [Queue](#) usage:

```

#include "myLibrary.h"
int main() {
    // Create an empty Queue of int type
    Queue queue1 = newQueue("%i");
    // Print queue1
    printQueue("%i\n", queue1);
    // Output:
    // Empty
    // Enqueue three items in queue1
    enqueue(queue1, 3);
    enqueue(queue1, 4);
    enqueue(queue1, -1);
    // Now queue1 contains: 3, 4, -1
    int extracted;
    // Dequeue the item on top from queue1 and save it into extracted
    dequeue(queue1, &extracted);
    // Now queue1 contains: 4, -1; extracted is now 3
    // Peek the item on top from queue1 and save it into extracted
    peekQueue(queue1, &extracted);
    // queue1 still contains: 4, -1 extracted is now 4
    int myArray[] = {23, 4, 65, -5, 12};
    // Create a Queue of ints from the static array myArray which contains 5 elements
    Queue queue2 = chooseNewQueueFromArray("%i", myArray, 5);
    // Now queue2 contains: 23, 4, 65, -5, 12
    printQueue("% i", queue1);
    // Output:
    // 23 4 65 -5 12
    // Check if queue1 and queue2 have equal contents
    byte areEqual = areQueuesEqual(queue1, queue2);
    // areListsEqual is now FALSE (See
    // [constants.h](https://catomaior.github.io/myLibrary/constants_8h.html) docs for its numeric value)
    // Delete an item from the top of queue2 without saving it
    deleteHeadFromQueue(queue2);
    // Now queue2 contains: 4, 65, -5, 12
    // Get queue2 length
    unsigned int queue2Length = getQueueLength(queue2);
    // queue2Length is now 4
}

```

```
// Delete queue1 and queue2
deleteQueue(queue1);
deleteQueue(queue2);
// Memory used by queue1 and queue2 is now freed. In order to avoid memory leaks deleting Queues before
// they go out of their scope is always good practice
return 0;
}
```

1.2.5 Array algorithms

This library contains some basic functions that implement some commonly used algorithms for arrays and matrix, such as linear searching or sorting. These functions are massively used inside the library itself, but they can be useful out of that context too.

Since these functions work with standard C static arrays, they always have its size and its type, specified using the `printf` convention, as parameters.

See `arrays.h` in the docs for all the details.

Here are some examples of their usage:

```
#include "myLibrary.h"
int main() {
    int myArray[] = {23, 45, 11, -23, -43, 43};
    // Sort myArray (which contains 6 items) using a bubbleSort algorithm
    chooseBubbleSortArr("%i", myArray, 6);
    // myArray now contains: -43, -23, 11, 23, 43, 45
    // Find the index of an item inside an array
    int index = chooseLinearSearchArr("%i", myArray, 6, 11);
    // index is now 2
    int myMatrix[][6] = {{23, 45, 11, -23, -43, 43},
                        {23, 45, 11, -23, -43, 43}};

    // Print myMatrix
    printMatrix("%4i", myMatrix, 2, 6);
    // Output is:
    // 23 45 11 -23 -43 43
    // 23 45 11 -23 -43 43
}
```

1.2.6 Strings

This library contains some basic functions for working with strings, such as getting a string of arbitrary size and saving it in memory, checking if it ends with a given substring, changing its last characters and getting a copy of it.

See `strings.h` in the docs for all the details.

Here are some examples of their usage:

```
#include "myLibrary.h"
int main() {
    // Get a string from command line and save it in myString (See
    // [types.h](https://catomaioir.github.io/myLibrary/types_8h.html) for details about string type)
    string myString = getString();
    // Assuming the user Typed "Test" and pressed enter, myString now is: "Test"
    // Check if myString ends with "st"
    int endsWithST = endsWith(myString, "st");
    // endsWithST is now TRUE (See [constants.h](https://catomaioir.github.io/myLibrary/constants_8h.html)
    // docs for its numeric value)
    // Create a new string with different last character from myString
    string newString = changeLastCharacter(myString, "T");
    // newString is now: "TeST"
    // Create a copy of newString
    string otherString = copyOf(newString);
    // newString is now: "TeST"
}
```

1.2.7 Miscellaneous

This library contains also standard comparing functions for `char`, `int`, `float`, `double` and pointer type and also two functions that try to allocate or reallocate memory. These functions are massively used inside the library itself, but they can be useful out of that context too.

See `utility.h` in the docs for all the details.

Here are some examples of their usage:

```
#include "myLibrary.h"
int main() {
    int a = 0, b = 1;
    // Compare a and b as integer values
    byte compare = chooseCmp("%i", &a, &b);
    // compare is now SMALLER (See [constants.h] (https://catomaior.github.io/myLibrary/constants\_8h.html)
    // docs for its numeric value)
    // Get a pointer to a dynamically allocated buffer of 1 byte
    void *ptr = saferMalloc(1);
    // ptr is now a pointer to a 1 byte buffer. If memory cannot be allocated the program prints the
    // following and exits:
    // An errorr occurred in function saferMalloc:
    // Could not allocate memory
    // Exiting
    // Resize an already allocated buffer
    ptr = saferRealloc(ptr, 2);
    // ptr is now a pointer to a 2 byte buffer. If memory cannot be reallocated the program prints the
    // following and exits:
    // An errorr occurred in function saferRealloc:
    // Could not reallocate memory
    // Exiting
}
```

1.3 How to import

1.3.1 On Linux

Download the build for Linux, unzip it and place it somewhere. Consider the following code:

```
#include "myLibrary.h"
int main() {
    byte myMatrix[][2] = {{42, 24}, {-24, 42}};
    printMatrix("%3hi", myMatrix, 2, 2);
    return 0;
}
```

Assuming it is saved in a file named `myFile.c` and you want to compile it using `gcc`, the correct command for compilation is:

```
gcc path/to/myFile.c -o path/to/myFileExecutable -I path/to/folder/with/myLibrary \
    path/to/folder/with/myLibrary/build/myLibrary_Linux.lib
```

Where:

- `path/to/myFile.c` is the relative or absolute path to `myFile.c`
- `path/to/myFileExecutable` is the relative or absolute path for the compiler output
- `path/to/folder/with/myLibrary` is the relative or absolute path of extracted `myLibrary` folder
- `path/to/folder/with/myLibrary/build/myLibrary_Linux.lib` is the path to the binary file of the library

1.3.2 On Visual Studio for Windows

Download the build for Windows, unzip it and place it somewhere. Steps to import:

- Open the solution where you want to use `myLibrary`
- Ensure the source file where you want to import `myLibrary` has `.c` extension. If its extension is `.cpp`, change it to `.c`
- Go to "Project" > "myProject Properties"

- In "Configuration" choose "All Configurations"
- In "Platform" choose "x64"
- Go to "Configuration Properties" > "C/C++" > "General". In "Additional Include Directories" add the path of the myLibrary folder you extracted before
- Go to "Configuration Properties" > "Linker" > "General". In "Additional Library Directories" add the path of the "build" folder inside the myLibrary folder you extracted before
- Go to "Configuration Properties" > "Linker" > "Input". In "Additional Dependencies" add `myLibrary_↵
Windows.lib; legacy_stdio_definitions.lib; legacy_stdio_wide_specifiers.↵
lib;`
- Click on "Ok" at the bottom of the window
- Near to "Local Windows Debugger" choose "x64". Now you are ready to `#include "myLibrary.h"` and compile and run your code

1.4 How to compile from source

Compilation from source is currently supported only on Linux. The only dependencies are `gcc` and `make`. Run:

```
git clone https://github.com/CatoMaior/myLibrary.git
cd myLibrary
make lib
```

The compiled binaries are `myLibrary_Linux.lib` and `myLibrary_Windows.lib` in the `build` folder.

If you want a pdf version of the docs too run:

```
make docs
```

The pdf is now in the `docs` folder

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

ArrayList		
	ArrayList type	13
LinkedList		
	LinkedList type	14
node		
	Node type	16
Queue		
	Queue type	17
Stack		
	Stack type	19

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	21
arrays.h	Common tasks with arrays: sorting, searching, printing etc	34
constants.h	Definition of symbolic constants used by the library	43
linkedList.h	Functions for working with LinkedList type	46
macros.h	Macros for emulated overloading	57
myLibrary.h	Includes all other headers. Useful for rapid import	74
queue.h	Functions for working with Queue type	75
stack.h	Functions for working with Stack type	84
strings.h	Common tasks with strings	92
types.h	Collection of useful types	95
utility.h	Common tasks such as comparing variables, allocate memory	97

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:

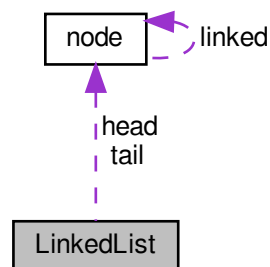
- [types.h](#)

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:

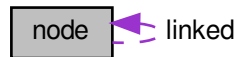
- [types.h](#)

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:

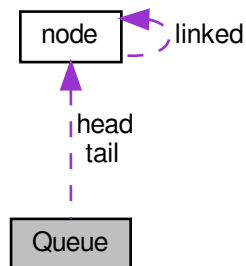
- [types.h](#)

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:

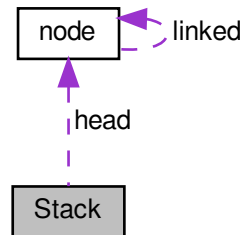
- [types.h](#)

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

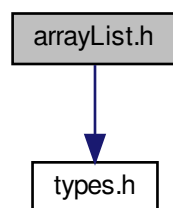
File Documentation

5.1 arrayList.h File Reference

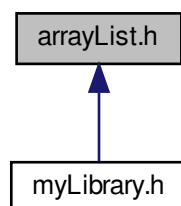
Functions for working with [ArrayList](#) type.

```
#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 ArrayList you want to append an item to
...	The item you want to append to <i>list</i>

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 ArrayList you want to compare
<i>list2</i>	The second ArrayList you want to compare
...	The comparison function needed to compare items inside given lists. This parameter is necessary only for pointer ArrayList type and is ignored otherwise. Must be a function that takes two pointers as argument and returns a positive int if the item pointed by the first argument is greater than the item pointed by the second argument, a negative int if the item pointed by the first argument is smaller than the item pointed by second, a zero int if the item pointed by first and second arguments are equal

Note

If comparing two pointer [ArrayList](#) type and the comparing function is not given a compiler nor runtime error is given, but the result of the comparison is unpredictable

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
...	The comparison function needed to compare items inside given lists. This parameter is necessary only for pointer ArrayList type and is ignored otherwise. Must be a function that takes two pointers as argument and returns a positive int if the item pointed by the first argument is greater than the item pointed by the second argument, a negative int if the item pointed by the first argument is smaller than the item pointed by second, a zero int if the item pointed by first and second arguments are equal

Note

If sorting an [ArrayList](#) type and the comparing function is not passed a compiler error is not given, but the [ArrayList](#) will be messed up

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>

Note

When creating an [ArrayList](#) from a pointer array the pointers are inserted into the [ArrayList](#), not what they point to

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
...	The function used to free memory pointed by every pointer of the ArrayList . Must be a function that takes a pointer as argument. Necessary only for pointer ArrayList type, ignored otherwise. When deleting a pointer ArrayList type if no free function is passed no compiler errors are thrown but you may cause severe memory leaks

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
<i>...</i>	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 [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. If searching in a pointer ArrayList type, after the item you want so search, you must provide the comparison function needed to compare the item you want to search and the items in the ArrayList . Must be a function that takes two pointers as argument and returns a zero int only if the item pointed by first and second arguments are equal

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. If searching in a pointer [ArrayList](#) type and the comparing function is not passed a compiler error is not given either, but the return value of the function can be unpredictable

Return values

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

5.1.2.11 `linearSearchAL()`

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

Linear search for [ArrayList](#).

Parameters

<i>list</i>	The ArrayList to be inspected
...	The key to be searched. If searching in a pointer ArrayList type, after the item you want to search, you must provide the comparison function needed to compare the item you want to search and the items in the ArrayList . Must be a function that takes two pointers as argument and returns a zero int only if the item pointed by first and second arguments are equal

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 item to be searched, still no errors occur but the return value of the function can be unpredictable. If searching in a pointer [ArrayList](#) type and the comparing function is not passed a compiler or runtime error is not given either, 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

<code>KEY_NOT_FOUND</code>	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
-------------	--

Note

When creating an [ArrayList](#) from a pointer [ArrayList](#) type the pointers in `list` are inserted into the [ArrayList](#), not what they point to

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
...	The comparison function needed to compare items inside given lists. This parameter is necessary only for pointer ArrayList type and is ignored otherwise. Must be a function that takes two pointers as argument and returns a positive int if the item pointed by the first argument is greater than the item pointed by the second argument, a negative int if the item pointed by the first argument is smaller than the item pointed by second, a zero int if the item pointed by first and second arguments are equal

Note

If sorting a pointer [ArrayList](#) type and the comparing function is not passed a compiler error is not given, but the [ArrayList](#) will be messed up

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
...	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 arrayList.h

[Go to the documentation of this file.](#)

```

1
7 #ifndef SEEN_ARRAYLIST
8 #define SEEN_ARRAYLIST
9
10 #include "types.h"
11
12 // TYPE INDEPENDENT FUNCTIONS
18 ArrayList newAL(const spec_t spec);
19
26 ArrayList newALFromAL(const ArrayList list);
27
34 void appendToAL(ArrayList list, ...);
35
43 void insertToAL(ArrayList list, unsigned int index, ...);
44
52 void setALItem(ArrayList list, unsigned int index, ...);
53
59 void mergeAL(ArrayList list1, const ArrayList list2);
60
67 void sliceAL(ArrayList list, unsigned int begin, unsigned int end);
68
74 void printAL(const spec_t spec, const ArrayList list);
75
81 void removeFromAL(ArrayList list, unsigned int index);
82
89 void getFromAL(const ArrayList list, unsigned int index, void *dest);
90
96 void deleteAL(ArrayList list, ...);
97
108 byte areALEqual(const ArrayList list1, const ArrayList list2, ...);
109
114 void reverseAL(ArrayList list);
115
122 void bubbleSortAL(ArrayList list, ...);
123
130 void quickSortAL(ArrayList list, ...);
131
140 byte isInAL(ArrayList list, ...);
141
150 int linearSearchAL(ArrayList list, ...);
151
160 ArrayList chooseNewALFromArray(const spec_t spec, const void *list, unsigned int size);
161
162 // TYPE DEPENDENT FUNCTIONS
167 ArrayList newALFromCharArray(const char list[], unsigned int size);
168
173 ArrayList newALFromArray(const char list[], unsigned int size);
174
179 ArrayList newALFromIntArray(const int list[], unsigned int size);
180
185 ArrayList newALFromFloatArray(const float list[], unsigned int size);
186
191 ArrayList newALFromDoubleArray(const double list[], unsigned int size);
192
197 ArrayList newALFromPtrArray(const void *list, unsigned int size);
198
204 unsigned int getALLength(const ArrayList list);
205
212 byte isEmptyAL(ArrayList list);
213
214 #endif

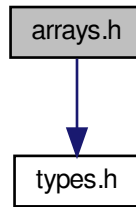
```

5.3 arrays.h File Reference

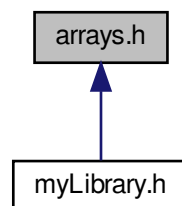
Common tasks with arrays: sorting, searching, printing etc.

```
#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 [chooseLinearSearchArr](#) (const [spec_t](#) spec, const 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 [charBubbleSortArr](#) (char *arr, unsigned int size)
Bubblesort for arrays of chars.
- void [intBubbleSortArr](#) (int *arr, unsigned int size)
Bubblesort for arrays of ints.
- void [floatBubbleSortArr](#) (float *arr, unsigned int size)
Bubblesort for arrays of floats.
- void [doubleBubbleSortArr](#) (double *arr, unsigned int size)

- Bubblesort for arrays of doubles.*
- void [ptrBubbleSortArr](#) (void **arr, unsigned int size, int(*cmpFunc)(const void *a, const void *b))
- Bubblesort for arrays of pointers.*
- void [charQuickSortArr](#) (char *arr, int size)
- Quicksort for arrays of chars.*
- void [intQuickSortArr](#) (int *arr, int size)
- Quicksort for arrays of ints.*
- void [floatQuickSortArr](#) (float *arr, int size)
- Quicksort for arrays of floats.*
- void [doubleQuickSortArr](#) (double *arr, int size)
- Quicksort for arrays of doubles.*
- void [ptrQuickSortArr](#) (void *arr, int size, int(*cmpFunc)(const void *a, const void *b))
- Quicksort for arrays of pointers.*
- int [charLinearSearchArr](#) (const char *arr, int size, char key)
- Linear search for arrays of chars.*
- int [intLinearSearchArr](#) (const char *arr, int size, int key)
- Linear search for arrays of integers.*
- int [floatLinearSearchArr](#) (const char *arr, int size, float key)
- Linear search for arrays of floats.*
- int [doubleLinearSearchArr](#) (const char *arr, int size, double key)
- Linear search for arrays of doubles.*
- int [ptrLinearSearchArr](#) (const void *arr, int size, void *key, int(*cmpFunc)(const void *a, const void *b))
- Linear search for arrays of pointers.*

5.3.1 Detailed Description

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

Author

Pietro Firpo (pietro.firpo@pm.me)

5.3.2 Function Documentation

5.3.2.1 [charBubbleSortArr\(\)](#)

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

Bubblesort for arrays of chars.

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

5.3.2.2 charLinearSearchArr()

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

Linear search for arrays of chars.

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

5.3.2.3 charQuickSortArr()

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

Quicksort for arrays of chars.

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

5.3.2.4 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
...	The comparison function needed to compare items inside given lists. This parameter is necessary only for pointer ArrayList type and is ignored otherwise. Must be a function that takes two pointers as argument and returns a positive int if the item pointed by the first argument is greater than the item pointed by the second argument, a negative int if the item pointed by the first argument is smaller than the item pointed by second, a zero int if the item pointed by first and second arguments are equal

5.3.2.5 chooseLinearSearchArr()

```
int chooseLinearSearchArr (
    const spec_t spec,
    const 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
...	The key to be searched. If searching in a pointer array, after the item you want to search, you must provide the comparison function needed to compare the item you want to search and the items in the array. Must be a function that takes two pointers as argument and returns a zero int only if the item pointed by first and second arguments are equal

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 item to be searched, still no errors occur but the return value of the function can be unpredictable. If searching in a pointer array and the comparing function is not passed a compiler or runtime error is not given either, 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
----------------------	-----------------------

5.3.2.6 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
...	The comparison function needed to compare items inside given lists. This parameter is necessary only for pointer ArrayList type and is ignored otherwise. Must be a function that takes two pointers as argument and returns a positive int if the item pointed by the first argument is greater than the item pointed by the second argument, a negative int if the item pointed by the first argument is smaller than the item pointed by second, a zero int if the item pointed by first and second arguments are equal

5.3.2.7 doubleBubbleSortArr()

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

Bubblesort for arrays of doubles.

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

5.3.2.8 doubleLinearSearchArr()

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

Linear search for arrays of doubles.

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

5.3.2.9 doubleQuickSortArr()

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

Quicksort for arrays of doubles.

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

5.3.2.10 floatBubbleSortArr()

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

Bubblesort for arrays of floats.

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

5.3.2.11 floatLinearSearchArr()

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

Linear search for arrays of floats.

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

5.3.2.12 floatQuickSortArr()

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

Quicksort for arrays of floats.

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

5.3.2.13 intBubbleSortArr()

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

Bubblesort for arrays of ints.

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

5.3.2.14 intLinearSearchArr()

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

Linear search for arrays of integers.

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

5.3.2.15 intQuickSortArr()

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

Quicksort for arrays of ints.

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

5.3.2.16 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 printf() 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

5.3.2.17 ptrBubbleSortArr()

```
void ptrBubbleSortArr (
    void ** arr,
    unsigned int size,
    int (*)(const void *a, const void *b) cmpFunc )
```

Bubblesort for arrays of pointers.

Parameters

<i>arr</i>	The array to be sorted
<i>size</i>	The number of items contained in arr
<i>cmpFunc</i>	The comparison function needed to compare items inside given lists. Must be a function that takes two pointers as argument and returns a positive int if the item pointed by the first argument is greater than the item pointed by the second argument, a negative int if the item pointed by the first argument is smaller than the item pointed by second, a zero int if the item pointed by first and second arguments are equal

5.3.2.18 ptrLinearSearchArr()

```
int ptrLinearSearchArr (
    const void * arr,
    int size,
    void * key,
    int (*)(const void *a, const void *b) cmpFunc )
```

Linear search for arrays of pointers.

Parameters

<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>key</i>	The key to be searched
<i>cmpFunc</i>	The comparison function to be used. Must be a function that returns a positive int if first argument is greater than the second, a negative byte if first argument is smaller than the second, a zero byte if first and second arguments are equal

Returns

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

Return values

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

5.3.2.19 ptrQuickSortArr()

```
void ptrQuickSortArr (
    void * arr,
    int size,
    int(*) (const void *a, const void *b) cmpFunc )
```

Quicksort for arrays of pointers.

Parameters

<i>arr</i>	The array to be sorted
<i>size</i>	The number of items contained in arr
<i>cmpFunc</i>	The comparison function to be used. Must be a function that returns a positive int if first argument is greater than the second, a negative byte if first argument is smaller than the second, a zero byte if first and second arguments are equal

5.4 arrays.h

[Go to the documentation of this file.](#)

```
1
7 #ifndef SEEN_ARRAYS
8 #define SEEN_ARRAYS
9
10 #include "types.h"
11
12 // Do the pointer version for ArrayLists, LinkedLists, Stacks, Queues
13
21 void chooseBubbleSortArr(const spec_t spec, void *arr, unsigned int size, ...);
22
30 void chooseQuickSortArr(const spec_t spec, void *arr, int size, ...);
31
42 int chooseLinearSearchArr(const spec_t spec, const void *arr, int size, ...);
```

```

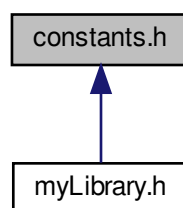
43
52 void printMatrix(const spec_t spec, const void *matrix, const unsigned int nRows, const unsigned int
    nColumns);
53
58 void charBubbleSortArr(char *arr, unsigned int size);
59
64 void intBubbleSortArr(int *arr, unsigned int size);
65
70 void floatBubbleSortArr(float *arr, unsigned int size);
71
76 void doubleBubbleSortArr(double *arr, unsigned int size);
77
84 void ptrBubbleSortArr(void **arr, unsigned int size, int (*cmpFunc)(const void *a, const void *b));
85
90 void charQuickSortArr(char *arr, int size);
91
96 void intQuickSortArr(int *arr, int size);
97
102 void floatQuickSortArr(float *arr, int size);
103
108 void doubleQuickSortArr(double *arr, int size);
109
116 void ptrQuickSortArr(void *arr, int size, int (*cmpFunc)(const void *a, const void *b));
117
122 int charLinearSearchArr(const char *arr, int size, char key);
123
128 int intLinearSearchArr(const char *arr, int size, int key);
129
134 int floatLinearSearchArr(const char *arr, int size, float key);
135
140 int doubleLinearSearchArr(const char *arr, int size, double key);
141
151 int ptrLinearSearchArr(const void *arr, int size, void *key, int (*cmpFunc)(const void *a, const void
    *b));
152
153 #endif

```

5.5 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.

5.5.1 Detailed Description

Definition of symbolic constants used by the library.

Author

Pietro Firpo (pietro.firpo@pm.me)

5.5.2 Macro Definition Documentation

5.5.2.1 EQUAL

```
#define EQUAL 0
```

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

5.5.2.2 FALSE

```
#define FALSE 0
```

Bool value definition.

5.5.2.3 GREATER

```
#define GREATER 1
```

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

5.5.2.4 KEY_NOT_FOUND

```
#define KEY_NOT_FOUND -1
```

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

5.5.2.5 SMALLER

```
#define SMALLER -1
```

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

5.5.2.6 TRUE

```
#define TRUE 0xFF
```

Bool value definition.

5.6 constants.h

[Go to the documentation of this file.](#)

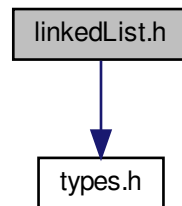
```
1
7 #ifndef SEEN_CONSTANTS
8 #define SEEN_CONSTANTS
9
13 #define GREATER 1
14
18 #define EQUAL 0
19
23 #define SMALLER -1
24
28 #define TRUE 0xFF
29
33 #define FALSE 0
34
38 #define KEY_NOT_FOUND -1
39
40 #endif
```

5.7 linkedList.h File Reference

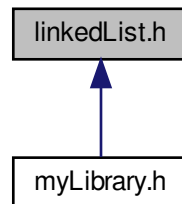
Functions for working with [LinkedList](#) type.

```
#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 `getLLLLength` (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.

5.7.1 Detailed Description

Functions for working with `LinkedList` type.

Author

Pietro Firpo (pietro.firpo@pm.me)

5.7.2 Function Documentation

5.7.2.1 appendToLL()

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

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

Parameters

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

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

5.7.2.2 appendToLLFromPtr()

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

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

Parameters

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

5.7.2.3 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

5.7.2.4 `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

5.7.2.5 `deleteLL()`

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

Delete a [LinkedList](#).

Parameters

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

5.7.2.6 `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

5.7.2.7 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`

5.7.2.8 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

5.7.2.9 `isInLL()`

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

Detect if an element is inside a `LinkedList`.

Parameters

<i>list</i>	The <code>LinkedList</code> 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>

5.7.2.10 `isLLEmpty()`

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

Check if `LinkedList` is empty.

Parameters

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

Return values

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

5.7.2.11 linearSearchLL()

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

Linear search for [LinkedList](#).

Parameters

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

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

5.7.2.12 linearSearchLLPtr()

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

Linear search for [LinkedList](#).

Parameters

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

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

5.7.2.13 `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

5.7.2.14 `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](#)

5.7.2.15 `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\(\)](#)

5.7.2.16 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\(\)](#)

5.7.2.17 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\(\)](#)

5.7.2.18 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\(\)](#)

5.7.2.19 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`

5.7.2.20 `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\(\)](#)

5.7.2.21 `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

5.7.2.22 `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

5.7.2.23 `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
<i>...</i>	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

5.7.2.24 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

5.8 linkedList.h

[Go to the documentation of this file.](#)

```
1
7 #ifndef SEEN_LINKEDLIST
8 #define SEEN_LINKEDLIST
9
10 #include "types.h"
11
17 LinkedList newLL(const spec\_t spec);
18
26 LinkedList chooseNewLLFromArray(const spec\_t spec, const void *arr, unsigned int size);
27
33 void printLL(const spec\_t spec, const LinkedList list);
34
41 void appendToLL(LinkedList list, ...);
42
48 void appendToLLFromPtr(LinkedList list, const void *element);
49
57 void insertToLL(LinkedList list, unsigned int index, ...);
58
63 void deleteLL(LinkedList list);
64
71 void getFromLL(LinkedList list, unsigned int index, void *dest);
72
80 void setLLItem(LinkedList list, unsigned int index, ...);
81
87 void removeFromLL(LinkedList list, unsigned int index);
88
```



```

94 void mergeLL(LinkedList list1, const LinkedList list2);
95
101 LinkedList newLLFromLL(const LinkedList list);
102
109 void sliceLL(LinkedList list, unsigned int begin, unsigned int end);
110
120 int linearSearchLL(LinkedList list, ...);
121
131 void *linearSearchLLPtr(LinkedList list, ...);
132
141 byte areLLEqual(const LinkedList list1, const LinkedList list2);
142
151 byte isInLL(LinkedList list, ...);
152
158 unsigned int getLLLength(const LinkedList list);
159
164 LinkedList newLLFromCharArray(const char arr[], unsigned int size);
165
170 LinkedList newLLFromArray(const int arr[], unsigned int size);
171
176 LinkedList newLLFromFloatArray(const float arr[], unsigned int size);
177
182 LinkedList newLLFromDoubleArray(const double arr[], unsigned int size);
183
188 LinkedList newLLFromPtrArray(const void *arr, unsigned int size);
189
196 byte isLLEmpty(LinkedList list);
197
198 // TODO Sorting algorithms, currently available only for ArrayList
199
200 #endif

```

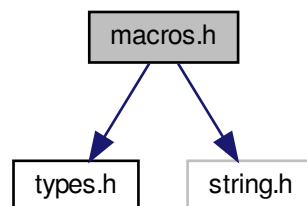
5.9 macros.h File Reference

Macros for emulated overloading.

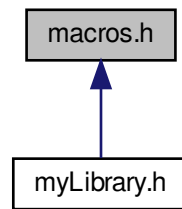
```
#include "types.h"
```

```
#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 [peek](#)(collection, dest)

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

5.9.1 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 `floatBubbleSortArr()` or `chooseBubbleSortArr()`. Moreover, these macros don't work with pointer arrays, [ArrayList](#), [LinkedList](#), [Stack](#) or [Queue](#) type

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

5.9.2 Macro Definition Documentation

5.9.2.1 `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 <code>list</code>

5.9.2.2 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

5.9.2.3 bubbleSortArr

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

Value:

```
_Generic(arr, char * \
: charBubbleSortArr, int * \
: intBubbleSortArr, float * \
: floatBubbleSortArr, double * \
: doubleBubbleSortArr, void ** \
: ptrBubbleSortArr)(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

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

5.9.2.5 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
-------------------	--

5.9.2.6 deleteHead

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

Value:

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

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

Parameters

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

5.9.2.7 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

5.9.2.8 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`

5.9.2.9 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>

5.9.2.10 isEmpty

```

#define isEmpty(
    collection )

```

Value:

```

_Generic(collection, ArrayList \
: isEmpty, LinkedList \
: isLLEmpty, Stack \
: isStackEmpty, 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>	collection is empty
<i>FALSE</i>	collection is not empty

5.9.2.11 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>

5.9.2.12 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

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

5.9.2.13 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

5.9.2.14 newALFromArray

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

Value:

```

_Generic(arr, char *
: newALFromCharArray, int *
: newALFromIntArray, float *
: newALFromFloatArray, double *
: newALFromDoubleArray, void **
: newALFromPtrArray)(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 <i>arr</i>

Returns

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

5.9.2.15 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 <i>arr</i>

Returns

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

5.9.2.16 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 <i>arr</i>

Returns

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

5.9.2.17 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 <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

5.9.2.18 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

5.9.2.19 peek

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

Value:

```
_Generic(list, Stack
: peekStack, Queue \
: peekQueue)(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

5.9.2.20 `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

5.9.2.21 `quickSortArr`

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

Value:

```
_Generic(arr, char *          \
: charQuickSortArr, int *      \
: intQuickSortArr, float *     \
: floatQuickSortArr, double *  \
: doubleQuickSortArr, void ** \
: ptrQuickSortArr)(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

5.9.2.22 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

5.9.2.23 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

5.9.2.24 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

5.10 macros.h

[Go to the documentation of this file.](#)

```
1
9 #ifndef SEEN_MACROS
10 #define SEEN_MACROS
11
12 #include "types.h"
13 #include <string.h>
14
24 #define cmpVal(a, b) _Generic((a, b), char * \
25 : charCmp, int * \
26 : intCmp, float * \
27 : floatCmp, double * \
28 : doubleCmp, void ** \
29 : ptrCmp)(a, b)
30
37 #define bubbleSortArr(arr, size) _Generic(arr, char * \
38 : charBubbleSortArr, int * \
39 : intBubbleSortArr, float * \
40 : floatBubbleSortArr, double * \
41 : doubleBubbleSortArr, void ** \
42 : ptrBubbleSortArr)(arr, size)
43
50 #define quickSortArr(arr, size, ...) _Generic(arr, char * \
51 : charQuickSortArr, int * \
52 : intQuickSortArr, float * \
53 : floatQuickSortArr, double * \
54 : doubleQuickSortArr, void ** \
55 : ptrQuickSortArr)(arr, size, ...)
56
63 #define newALFromArray(arr, size) _Generic(arr, char * \
64 : newALFromCharArray, int * \
65 : newALFromIntArray, float * \
66 : newALFromFloatArray, double * \
67 : newALFromDoubleArray, void ** \
68 : newALFromPtrArray)(arr, size)
69
76 #define newLLFromArray(arr, size) _Generic(arr, char * \
77 : newLLFromCharArray, int * \
78 : newLLFromIntArray, float * \
79 : newLLFromFloatArray, double * \
80 : newLLFromDoubleArray, void ** \
81 : newLLFromPtrArray)(arr, size)
82
89 #define newStackFromArray(arr, size) _Generic(arr, char * \
90 : newStackFromCharArray, int * \
91 : newStackFromIntArray, float * \
92 : newStackFromFloatArray, double * \
93 : newStackFromDoubleArray, void ** \
94 : newStackFromPtrArray)(arr, size)
95
102 #define newQueueFromArray(arr, size) _Generic(arr, char * \
103 : newQueueFromCharArray, int * \
104 : newQueueFromIntArray, float * \
105 : newQueueFromFloatArray, double * \
106 : newQueueFromDoubleArray, void ** \
107 : newQueueFromPtrArray)(arr, size)
108
115 #define newStackFromArray(arr, size) _Generic(arr, char * \
116 : newStackFromCharArray, int * \
117 : newStackFromIntArray, float * \
```



```

118                                     : newStackFromFloatArray, double * \
119                                     : newStackFromDoubleArray, void ** \
120                                     : newStackFromPtrArray)(arr, size)
121
122 #define print(spec, collection) _Generic(collection, ArrayList      \
123                                     : printAL, LinkedList \
124                                     : printLL, Stack      \
125                                     : printStack, Queue   \
126                                     : printQueue)(spec, collection)
127
128 #define areEqual(collection1, collection2) _Generic(collection1, ArrayList      \
129                                     : areALEqual, LinkedList \
130                                     : areLLEqual, Stack      \
131                                     : areStacksEqual, Queue   \
132                                     : areQueuesEqual)(collection1, collection2)
133
134 #define append(list, item) _Generic(list, ArrayList      \
135                                     : appendToAL, LinkedList \
136                                     : appendToLL)(list, item)
137
138 #define insert(list, index, item) _Generic(list, ArrayList      \
139                                     : insertToAL, LinkedList \
140                                     : insertToLL)(list, index, item)
141
142 #define set(list, index, newItem) _Generic(list, ArrayList      \
143                                     : setALItem, LinkedList \
144                                     : setLLItem)(list, index, newItem)
145
146 #define merge(list1, list2) _Generic(list1, ArrayList      \
147                                     : mergeAL, LinkedList \
148                                     : mergeLL)(list1, list2)
149
150 #define slice(list, begin, end) _Generic(list, ArrayList      \
151                                     : sliceAL, LinkedList \
152                                     : sliceLL)(list, begin, end)
153
154 #define removeItem(list, index) _Generic(list, ArrayList      \
155                                     : removeFromAL, LinkedList \
156                                     : removeFromLL)(list, index)
157
158 #define getItem(list, index, dest) _Generic(list, ArrayList      \
159                                     : getFromAL, LinkedList \
160                                     : getFromLL)(list, index, dest)
161
162 #define delete(collection) _Generic(collection, ArrayList      \
163                                     : deleteAL, LinkedList \
164                                     : deleteLL, Stack      \
165                                     : deleteStack, Queue   \
166                                     : deleteQueue)(collection)
167
168 #define isIn(collection, item) _Generic(collection, ArrayList      \
169                                     : isInAL, LinkedList \
170                                     : isInLL, Stack      \
171                                     : isInStack, Queue   \
172                                     : isInQueue)(collection, item)
173
174 #define getLength(collection) _Generic(collection, ArrayList      \
175                                     : getALLength, LinkedList \
176                                     : getLLLength, Stack      \
177                                     : getStackLength, Queue   \
178                                     : getQueueLength, string \
179                                     : strlen)(collection)
180
181 #define linearSearch(list, key) _Generic(list, ArrayList      \
182                                     : linearSearchAL, LinkedList \
183                                     : linearSearchLL)(list, key)
184
185 #define deleteHead(collection) _Generic(list, Stack      \
186                                     : deleteHeadFromStack, Queue \
187                                     : deleteHeadFromQueue)(collection)
188
189 #define isEmpty(collection) _Generic(collection, ArrayList      \
190                                     : isALEmpty, LinkedList \
191                                     : isLLEmpty, Stack      \
192                                     : isStackEmpty, Queue   \
193                                     : isQueueEmpty)(collection, item)
194
195 #define peek(collection, dest) _Generic(list, Stack      \
196                                     : peekStack, Queue \
197                                     : peekQueue)(collection)
198 #endif

```

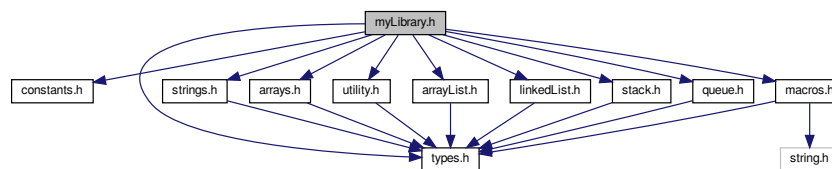
5.11 mainPage.md File Reference

5.12 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:



5.12.1 Detailed Description

Includes all other headers. Useful for rapid import.

Author

Pietro Firpo (pietro.firpo@pm.me)

5.13 myLibrary.h

[Go to the documentation of this file.](#)

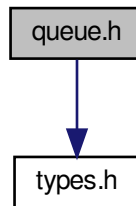
```
1
7 #include "constants.h"
8
9 #include "macros.h"
10
11 #include "types.h"
12
13 #include "strings.h"
14
15 #include "arrays.h"
16
17 #include "utility.h"
18
19 #include "arrayList.h"
20
21 #include "linkedList.h"
22
23 #include "stack.h"
24
25 #include "queue.h"
```

5.14 queue.h File Reference

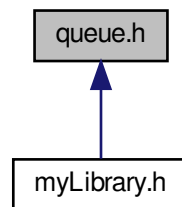
Functions for working with [Queue](#) type.

```
#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 `peekQueue` (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 `isEmpty` (`Stack` stack)
Check if `Queue` is empty.
- `Queue` `newQueueFromCharArray` (const char arr[], unsigned int size)
Create a `Queue` from an array of chars.
- `Queue` `newQueueFromArray` (const int arr[], unsigned int size)
Create a `Queue` from an array of integers.
- `Queue` `newQueueFromFloatArray` (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`.

5.14.1 Detailed Description

Functions for working with `Queue` type.

Author

Pietro Firpo (pietro.firpo@pm.me)

5.14.2 Function Documentation

5.14.2.1 `areQueuesEqual()`

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

Compare two `Queue`.

Parameters

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

5.14.2.2 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

5.14.2.3 deleteHeadFromQueue()

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

Delete current [Queue](#) head.

Parameters

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

5.14.2.4 deleteQueue()

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

Delete a [Queue](#).

Parameters

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

5.14.2.5 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

5.14.2.6 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

5.14.2.7 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>

5.14.2.8 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*

5.14.2.9 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>

5.14.2.10 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

5.14.2.11 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](#)

5.14.2.12 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\(\)](#)

5.14.2.13 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\(\)](#)

5.14.2.14 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\(\)](#)

5.14.2.15 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\(\)](#)

5.14.2.16 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\(\)](#)

5.14.2.17 peekQueue()

```
void peekQueue (
    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

5.14.2.18 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

5.15 queue.h

[Go to the documentation of this file.](#)

```
1
7 #ifndef SEEN_QUEUE
8 #define SEEN_QUEUE
9
10 #include "types.h"
11
17 Queue newQueue(const spec_t spec);
18
25 void enqueue(Queue queue, ...);
26
32 void dequeue(Queue queue, void *dest);
33
39 void printQueue(const spec_t spec, const Queue queue);
40
46 unsigned int getQueueLength(const Queue queue);
47
52 void deleteHeadFromQueue(Queue queue);
53
59 void peekQueue(const Queue queue, void *dest);
60
65 void deleteQueue(Queue queue);
66
76 byte isInQueue(Queue queue, ...);
77
85 Queue chooseNewQueueFromArray(const spec_t spec, const void *arr, unsigned int size);
86
92 void enqueueFromPtr(Queue queue, const void *element);
93
100 byte isEmptyQueue(Stack stack);
101
106 Queue newQueueFromCharArray(const char arr[], unsigned int size);
107
112 Queue newQueueFromArray(const int arr[], unsigned int size);
113
118 Queue newQueueFromFloatArray(const float arr[], unsigned int size);
119
124 Queue newQueueFromDoubleArray(const double arr[], unsigned int size);
125
130 Queue newQueueFromPtrArray(const void *arr, unsigned int size);
131
```

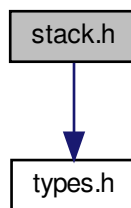
```
140 byte areQueuesEqual(const Queue queue1, const Queue queue2);  
141  
142 #endif
```

5.16 stack.h File Reference

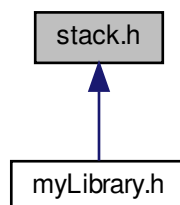
Functions for working with [Stack](#) type.

```
#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 `isEmpty` (`Stack` stack)
Check if `Stack` is empty.
- void `deleteStack` (`Stack` stack)
Delete a `Stack`.
- void `peekStack` (`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`.

5.16.1 Detailed Description

Functions for working with `Stack` type.

Author

Pietro Firpo (pietro.firpo@pm.me)

5.16.2 Function Documentation

5.16.2.1 `areStacksEqual()`

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

Compare two `Stack`.

Parameters

<i>stack1</i>	The first Stack you want to compare
<i>stack2</i>	The second Stack you want to compare

Returns

The result of the comparison

Return values

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

5.16.2.2 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 <i>arr</i>

Returns

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

5.16.2.3 deleteHeadFromStack()

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

Delete current [Stack](#) head.

Parameters

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

5.16.2.4 deleteStack()

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

Delete a [Stack](#).

Parameters

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

5.16.2.5 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*

5.16.2.6 isInStack()

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

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

Parameters

<i>stack</i>	The Stack you want search in
...	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 <code>stack</code>
<i>FALSE</i>	Given element is not contained in <code>stack</code>

5.16.2.7 isStackEmpty()

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

Check if [Stack](#) is empty.

Parameters

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

Return values

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

5.16.2.8 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](#)

5.16.2.9 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\(\)](#)

5.16.2.10 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\(\)](#)

5.16.2.11 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\(\)](#)

5.16.2.12 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\(\)](#)

5.16.2.13 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\(\)](#)

5.16.2.14 peekStack()

```
void peekStack (
    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

5.16.2.15 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

5.16.2.16 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

5.16.2.17 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

5.16.2.18 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>

5.17 stack.h

[Go to the documentation of this file.](#)

```
1
2
3 #ifndef SEEN_STACK
4 #define SEEN_STACK
5
6 #include "types.h"
7
8 Stack newStack(const spec\_t spec);
9
10 void push(Stack stack, ...);
11
12 void printStack(const spec\_t spec, const Stack stack);
13
14 void pop(Stack stack, void *dest);
15
16 void deleteHeadFromStack(Stack stack);
17
18 byte isEmptyStack(Stack stack);
19
20 void deleteStack(Stack stack);
21
22 void peekStack(Stack stack, void *dest);
23
24 byte isInStack(Stack stack, ...);
25
26 Stack chooseNewStackFromArray(const spec\_t spec, const void *arr, unsigned int size);
27
28 void pushFromPtr(Stack stack, const void *element);
29
30 unsigned int getStackLength(const Stack stack);
31
32 Stack newStackFromCharArray(const char arr[], unsigned int size);
```

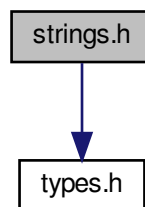
```
107
112 Stack newStackFromIntArray(const int arr[], unsigned int size);
113
118 Stack newStackFromFloatArray(const float arr[], unsigned int size);
119
124 Stack newStackFromDoubleArray(const double arr[], unsigned int size);
125
130 Stack newStackFromPtrArray(const void *arr, unsigned int size);
131
140 byte areStacksEqual(const Stack stack1, const Stack stack2);
141
142 #endif
```

5.18 strings.h File Reference

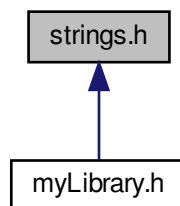
Common tasks with strings.

```
#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*.

5.18.1 Detailed Description

Common tasks with strings.

Author

Pietro Firpo (pietro.firpo@pm.me)

5.18.2 Function Documentation

5.18.2.1 changeLastCharacter()

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

Get a tring with different last character.

Parameters

<i>str</i>	The <i>string</i> 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
-------------	--

5.18.2.2 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](#)

5.18.2.3 endsWith()

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

Check if a [string](#) ends with the specified substrng.

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

5.18.2.4 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
-------------	--

5.19 strings.h

[Go to the documentation of this file.](#)

```

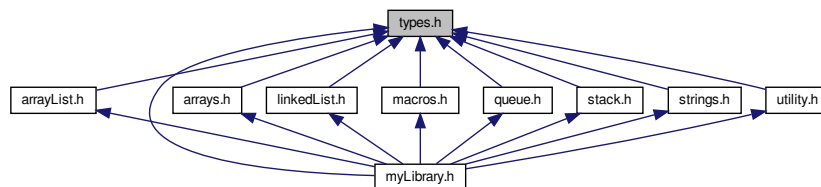
1
7 #ifndef SEEN_STRINGS
8 #define SEEN_STRINGS
9
10 #include "types.h"
11
17 string getString();
18
27 byte endsWith(const string str, const string suffix);
28
36 string changeLastCharacter(const string str, char newCharacter);
37
43 string copyOf(const string src);
44
45 #endif

```

5.20 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

5.20.1 Detailed Description

Collection of useful types.

Author

Pietro Firpo (pietro.firpo@pm.me)

5.20.2 Typedef Documentation

5.20.2.1 byte

```
typedef char byte
```

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

5.20.2.2 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

5.20.2.3 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

5.20.2.4 string

```
typedef char* string
```

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

5.21 types.h

[Go to the documentation of this file.](#)

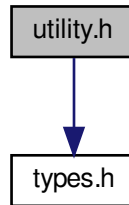
```
1
7 #ifndef SEEN_TYPES
8 #define SEEN_TYPES
9
13 typedef char byte;
14
20 typedef char *spec_t;
21
25 typedef char *string;
26
31 typedef struct {
35     spec_t type;
36
40     void *body;
41
45     unsigned int size;
46 } *ArrayList;
47
53 typedef struct node {
57     void *data;
58
62     struct node *linked;
63 } *Node;
64
69 typedef struct {
73     spec_t type;
74
78     Node head;
79
83     Node tail;
84
88     unsigned int size;
89 } *LinkedList;
90
95 typedef struct {
99     spec_t type;
100
104     Node head;
105 } *Stack;
106
111 typedef struct {
115     spec_t type;
116
120     Node head;
121
125     Node tail;
126
130     unsigned int size;
131 } *Queue;
132
133 #endif
```

5.22 utility.h File Reference

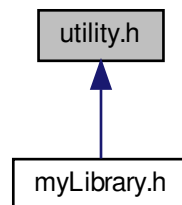
Common tasks such as comparing variables, allocate memory.

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

Include dependency graph for utility.h:



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



Functions

- int [chooseCmp](#) (const [spec_t](#) spec, const void *a, const void *b)
Compare two values.
- int [charCmp](#) (const void *a, const void *b)
Compare two chars.
- int [byteCmp](#) (const void *a, const void *b)
Compare two bytes.
- int [intCmp](#) (const void *a, const void *b)
Compare two ints.
- int [floatCmp](#) (const void *a, const void *b)
Compare two floats.
- int [doubleCmp](#) (const void *a, const void *b)
Compare two doubles.
- int [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.

5.22.1 Detailed Description

Common tasks such as comparing variables, allocate memory.

Author

Pietro Firpo (pietro.firpo@pm.me)

5.22.2 Function Documentation

5.22.2.1 byteCmp()

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

Compare two bytes.

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

5.22.2.2 charCmp()

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

Compare two chars.

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

5.22.2.3 chooseCmp()

```
int chooseCmp (
    const spec\_t spec,
    const void * a,
    const void * b )
```

Compare two values.

Parameters

<i>spec</i>	Type specifier of the values to be compared. 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 greater than the second
<i>EQUAL</i>	First element is equal to the second
<i>SMALLER</i>	First element is smaller than the second

5.22.2.4 doubleCmp()

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

Compare two doubles.

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

5.22.2.5 floatCmp()

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

Compare two floats.

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

5.22.2.6 intCmp()

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

Compare two ints.

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

5.22.2.7 ptrCmp()

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

Compare two pointers.

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

5.22.2.8 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

5.22.2.9 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

5.23 utility.h

[Go to the documentation of this file.](#)

```
1
7 #ifndef SEEN_UTILITY
8 #define SEEN_UTILITY
9
10 #include "types.h"
11
22 int chooseCmp(const spec_t spec, const void *a, const void *b);
23
28 int charCmp(const void *a, const void *b);
29
34 int byteCmp(const void *a, const void *b);
35
40 int intCmp(const void *a, const void *b);
41
46 int floatCmp(const void *a, const void *b);
47
52 int doubleCmp(const void *a, const void *b);
53
58 int ptrCmp(const void *a, const void *b);
59
66 void *saferMalloc(unsigned int bytes);
67
75 void *saferRealloc(void *pointer, unsigned int bytes);
76
77 #endif
```


Index

- append
 - macros.h, 59
- appendToAL
 - arrayList.h, 23
- appendToLL
 - linkedList.h, 47
- appendToLLFromPtr
 - linkedList.h, 48
- areALEqual
 - arrayList.h, 23
- areEqual
 - macros.h, 60
- areLLEqual
 - linkedList.h, 48
- areQueuesEqual
 - queue.h, 76
- areStacksEqual
 - stack.h, 85
- ArrayList, 13
 - body, 13
 - size, 13
 - type, 14
- arrayList.h, 21
 - appendToAL, 23
 - areALEqual, 23
 - bubbleSortAL, 24
 - chooseNewALFromArray, 24
 - deleteAL, 25
 - getALLength, 25
 - getFromAL, 26
 - insertToAL, 26
 - isALEmpty, 26
 - isInAL, 28
 - linearSearchAL, 28
 - mergeAL, 29
 - newAL, 29
 - newALFromAL, 30
 - newALFromByteArray, 30
 - newALFromCharArray, 30
 - newALFromDoubleArray, 30
 - newALFromFloatArray, 31
 - newALFromIntArray, 31
 - newALFromPtrArray, 31
 - printAL, 31
 - quickSortAL, 32
 - removeFromAL, 32
 - reverseAL, 32
 - setALItem, 33
 - sliceAL, 33
- arrays.h, 34
 - charBubbleSortArr, 36
 - charLinearSearchArr, 36
 - charQuickSortArr, 37
 - chooseBubbleSortArr, 37
 - chooseLinearSearchArr, 37
 - chooseQuickSortArr, 38
 - doubleBubbleSortArr, 39
 - doubleLinearSearchArr, 39
 - doubleQuickSortArr, 39
 - floatBubbleSortArr, 39
 - floatLinearSearchArr, 39
 - floatQuickSortArr, 39
 - intBubbleSortArr, 40
 - intLinearSearchArr, 40
 - intQuickSortArr, 40
 - printMatrix, 40
 - ptrBubbleSortArr, 41
 - ptrLinearSearchArr, 41
 - ptrQuickSortArr, 42
- body
 - ArrayList, 13
- bubbleSortAL
 - arrayList.h, 24
- bubbleSortArr
 - macros.h, 60
- byte
 - types.h, 96
- byteCmp
 - utility.h, 99
- changeLastCharacter
 - strings.h, 93
- charBubbleSortArr
 - arrays.h, 36
- charCmp
 - utility.h, 99
- charLinearSearchArr
 - arrays.h, 36
- charQuickSortArr
 - arrays.h, 37
- chooseBubbleSortArr
 - arrays.h, 37
- chooseCmp
 - utility.h, 99
- chooseLinearSearchArr
 - arrays.h, 37
- chooseNewALFromArray
 - arrayList.h, 24

- chooseNewLLFromArray
 - linkedList.h, [49](#)
- chooseNewQueueFromArray
 - queue.h, [77](#)
- chooseNewStackFromArray
 - stack.h, [86](#)
- chooseQuickSortArr
 - arrays.h, [38](#)
- cmpVal
 - macros.h, [61](#)
- constants.h, [43](#)
 - EQUAL, [44](#)
 - FALSE, [44](#)
 - GREATER, [44](#)
 - KEY_NOT_FOUND, [44](#)
 - SMALLER, [45](#)
 - TRUE, [45](#)
- copyOf
 - strings.h, [93](#)
- data
 - node, [16](#)
- delete
 - macros.h, [61](#)
- deleteAL
 - arrayList.h, [25](#)
- deleteHead
 - macros.h, [62](#)
- deleteHeadFromQueue
 - queue.h, [77](#)
- deleteHeadFromStack
 - stack.h, [86](#)
- deleteLL
 - linkedList.h, [49](#)
- deleteQueue
 - queue.h, [77](#)
- deleteStack
 - stack.h, [87](#)
- dequeue
 - queue.h, [78](#)
- doubleBubbleSortArr
 - arrays.h, [39](#)
- doubleCmp
 - utility.h, [100](#)
- doubleLinearSearchArr
 - arrays.h, [39](#)
- doubleQuickSortArr
 - arrays.h, [39](#)
- endsWith
 - strings.h, [94](#)
- enqueue
 - queue.h, [78](#)
- enqueueFromPtr
 - queue.h, [78](#)
- EQUAL
 - constants.h, [44](#)
- FALSE
 - constants.h, [44](#)
- floatBubbleSortArr
 - arrays.h, [39](#)
- floatCmp
 - utility.h, [100](#)
- floatLinearSearchArr
 - arrays.h, [39](#)
- floatQuickSortArr
 - arrays.h, [39](#)
- getALLength
 - arrayList.h, [25](#)
- getFromAL
 - arrayList.h, [26](#)
- getFromLL
 - linkedList.h, [49](#)
- getItem
 - macros.h, [62](#)
- getLength
 - macros.h, [62](#)
- getLLLength
 - linkedList.h, [50](#)
- getQueueLength
 - queue.h, [80](#)
- getStackLength
 - stack.h, [87](#)
- getString
 - strings.h, [94](#)
- GREATER
 - constants.h, [44](#)
- head
 - LinkedList, [15](#)
 - Queue, [18](#)
 - Stack, [19](#)
- insert
 - macros.h, [63](#)
- insertToAL
 - arrayList.h, [26](#)
- insertToLL
 - linkedList.h, [50](#)
- intBubbleSortArr
 - arrays.h, [40](#)
- intCmp
 - utility.h, [100](#)
- intLinearSearchArr
 - arrays.h, [40](#)
- intQuickSortArr
 - arrays.h, [40](#)
- isEmpty
 - arrayList.h, [26](#)
- isEmpty
 - macros.h, [63](#)
- isIn
 - macros.h, [64](#)
- isInAL
 - arrayList.h, [28](#)
- isInLL

- linkedList.h, 51
- isInQueue
 - queue.h, 80
- isInStack
 - stack.h, 87
- isLLEmpty
 - linkedList.h, 51
- isQueueEmpty
 - queue.h, 81
- isStackEmpty
 - stack.h, 88
- KEY_NOT_FOUND
 - constants.h, 44
- linearSearch
 - macros.h, 64
- linearSearchAL
 - arrayList.h, 28
- linearSearchLL
 - linkedList.h, 51
- linearSearchLLPtr
 - linkedList.h, 52
- linked
 - node, 16
- LinkedList, 14
 - head, 15
 - size, 15
 - tail, 15
 - type, 15
- linkedList.h, 46
 - appendToLL, 47
 - appendToLLFromPtr, 48
 - areLLEqual, 48
 - chooseNewLLFromArray, 49
 - deleteLL, 49
 - getFromLL, 49
 - getLLLength, 50
 - insertToLL, 50
 - isInLL, 51
 - isLLEmpty, 51
 - linearSearchLL, 51
 - linearSearchLLPtr, 52
 - mergeLL, 53
 - newLL, 53
 - newLLFromCharArray, 53
 - newLLFromDoubleArray, 53
 - newLLFromFloatArray, 54
 - newLLFromIntArray, 54
 - newLLFromLL, 54
 - newLLFromPtrArray, 54
 - printLL, 55
 - removeFromLL, 55
 - setLLItem, 55
 - sliceLL, 56
- macros.h, 57
 - append, 59
 - areEqual, 60
 - bubbleSortArr, 60
 - cmpVal, 61
 - delete, 61
 - deleteHead, 62
 - getItem, 62
 - getLength, 62
 - insert, 63
 - isEmpty, 63
 - isIn, 64
 - linearSearch, 64
 - merge, 66
 - newALFromArray, 66
 - newLLFromArray, 67
 - newQueueFromArray, 67
 - newStackFromArray, 68, 69
 - peek, 69
 - print, 70
 - quickSortArr, 70
 - removeItem, 71
 - set, 71
 - slice, 71
- mainPage.md, 74
- merge
 - macros.h, 66
- mergeAL
 - arrayList.h, 29
- mergeLL
 - linkedList.h, 53
- myLibrary.h, 74
- newAL
 - arrayList.h, 29
- newALFromAL
 - arrayList.h, 30
- newALFromArray
 - macros.h, 66
- newALFromByteArray
 - arrayList.h, 30
- newALFromCharArray
 - arrayList.h, 30
- newALFromDoubleArray
 - arrayList.h, 30
- newALFromFloatArray
 - arrayList.h, 31
- newALFromIntArray
 - arrayList.h, 31
- newALFromPtrArray
 - arrayList.h, 31
- newLL
 - linkedList.h, 53
- newLLFromArray
 - macros.h, 67
- newLLFromCharArray
 - linkedList.h, 53
- newLLFromDoubleArray
 - linkedList.h, 53
- newLLFromFloatArray
 - linkedList.h, 54
- newLLFromIntArray

- linkedList.h, [54](#)
- newLLFromLL
 - linkedList.h, [54](#)
- newLLFromPtrArray
 - linkedList.h, [54](#)
- newQueue
 - queue.h, [81](#)
- newQueueFromArray
 - macros.h, [67](#)
- newQueueFromCharArray
 - queue.h, [81](#)
- newQueueFromDoubleArray
 - queue.h, [81](#)
- newQueueFromFloatArray
 - queue.h, [82](#)
- newQueueFromIntArray
 - queue.h, [82](#)
- newQueueFromPtrArray
 - queue.h, [82](#)
- newStack
 - stack.h, [88](#)
- newStackFromArray
 - macros.h, [68](#), [69](#)
- newStackFromCharArray
 - stack.h, [88](#)
- newStackFromDoubleArray
 - stack.h, [89](#)
- newStackFromFloatArray
 - stack.h, [89](#)
- newStackFromIntArray
 - stack.h, [89](#)
- newStackFromPtrArray
 - stack.h, [89](#)
- Node
 - types.h, [96](#)
- node, [16](#)
 - data, [16](#)
 - linked, [16](#)
- peek
 - macros.h, [69](#)
- peekQueue
 - queue.h, [82](#)
- peekStack
 - stack.h, [89](#)
- pop
 - stack.h, [90](#)
- print
 - macros.h, [70](#)
- printAL
 - arrayList.h, [31](#)
- printLL
 - linkedList.h, [55](#)
- printMatrix
 - arrays.h, [40](#)
- printQueue
 - queue.h, [83](#)
- printStack
 - stack.h, [90](#)
- ptrBubbleSortArr
 - arrays.h, [41](#)
- ptrCmp
 - utility.h, [100](#)
- ptrLinearSearchArr
 - arrays.h, [41](#)
- ptrQuickSortArr
 - arrays.h, [42](#)
- push
 - stack.h, [90](#)
- pushFromPtr
 - stack.h, [91](#)
- Queue, [17](#)
 - head, [18](#)
 - size, [18](#)
 - tail, [18](#)
 - type, [18](#)
- queue.h, [75](#)
 - areQueuesEqual, [76](#)
 - chooseNewQueueFromArray, [77](#)
 - deleteHeadFromQueue, [77](#)
 - deleteQueue, [77](#)
 - dequeue, [78](#)
 - enqueue, [78](#)
 - enqueueFromPtr, [78](#)
 - getQueueLength, [80](#)
 - isInQueue, [80](#)
 - isEmpty, [81](#)
 - newQueue, [81](#)
 - newQueueFromCharArray, [81](#)
 - newQueueFromDoubleArray, [81](#)
 - newQueueFromFloatArray, [82](#)
 - newQueueFromIntArray, [82](#)
 - newQueueFromPtrArray, [82](#)
 - peekQueue, [82](#)
 - printQueue, [83](#)
- quickSortAL
 - arrayList.h, [32](#)
- quickSortArr
 - macros.h, [70](#)
- removeFromAL
 - arrayList.h, [32](#)
- removeFromLL
 - linkedList.h, [55](#)
- removeItem
 - macros.h, [71](#)
- reverseAL
 - arrayList.h, [32](#)
- saferMalloc
 - utility.h, [100](#)
- saferRealloc
 - utility.h, [101](#)
- set
 - macros.h, [71](#)
- setALItem
 - arrayList.h, [33](#)

- setLLItem
 - linkedList.h, 55
- size
 - ArrayList, 13
 - LinkedList, 15
 - Queue, 18
- slice
 - macros.h, 71
- sliceAL
 - arrayList.h, 33
- sliceLL
 - linkedList.h, 56
- SMALLER
 - constants.h, 45
- spec_t
 - types.h, 96
- Stack, 19
 - head, 19
 - type, 19
- stack.h, 84
 - areStacksEqual, 85
 - chooseNewStackFromArray, 86
 - deleteHeadFromStack, 86
 - deleteStack, 87
 - getStackLength, 87
 - isInStack, 87
 - isStackEmpty, 88
 - newStack, 88
 - newStackFromCharArray, 88
 - newStackFromDoubleArray, 89
 - newStackFromFloatArray, 89
 - newStackFromIntArray, 89
 - newStackFromPtrArray, 89
 - peekStack, 89
 - pop, 90
 - printStack, 90
 - push, 90
 - pushFromPtr, 91
- string
 - types.h, 96
- strings.h, 92
 - changeLastCharacter, 93
 - copyOf, 93
 - endsWith, 94
 - getString, 94
- tail
 - LinkedList, 15
 - Queue, 18
- TRUE
 - constants.h, 45
- type
 - ArrayList, 14
 - LinkedList, 15
 - Queue, 18
 - Stack, 19
- types.h, 95
 - byte, 96
 - Node, 96
 - spec_t, 96
 - string, 96
- utility.h, 97
 - byteCmp, 99
 - charCmp, 99
 - chooseCmp, 99
 - doubleCmp, 100
 - floatCmp, 100
 - intCmp, 100
 - ptrCmp, 100
 - saferMalloc, 100
 - saferRealloc, 101