

# C Data Structures Library

1.0.0

Generated by Doxygen 1.9.6



<b>1 Data Structure Library</b>	<b>1</b>
1.1 Linked List	1
1.2 Stack	1
1.3 Queue	1
1.4 What Data Type?	1
1.4.1 Documentation	2
1.4.2 Installation	2
1.4.3 Authors	2
1.4.4 Feedback	2
<b>2 Data Structure Index</b>	<b>3</b>
2.1 Data Structures	3
<b>3 File Index</b>	<b>5</b>
3.1 File List	5
<b>4 Data Structure Documentation</b>	<b>7</b>
4.1 LList Struct Reference	7
4.1.1 Detailed Description	7
4.2 Node Struct Reference	8
4.2.1 Detailed Description	8
4.3 Queue Struct Reference	8
4.3.1 Detailed Description	9
4.4 Stack Struct Reference	9
4.4.1 Detailed Description	9
<b>5 File Documentation</b>	<b>11</b>
5.1 Include/LinkedList.h File Reference	11
5.1.1 Detailed Description	12
5.1.2 Typedef Documentation	12
5.1.2.1 compare	13
5.1.2.2 LIST	13
5.1.2.3 NODE	13
5.1.3 Function Documentation	13
5.1.3.1 Add()	13
5.1.3.2 DestroyList()	14
5.1.3.3 DumpList()	14
5.1.3.4 findMid()	15
5.1.3.5 Get()	15
5.1.3.6 IndexOf()	16
5.1.3.7 InitList()	17
5.1.3.8 InsertNodeAfterTarget()	18
5.1.3.9 InsertNodeBeforeTarget()	18
5.1.3.10 MergeSort()	19

5.1.3.11 RemoveByIndex()	20
5.1.3.12 Sort()	21
5.1.3.13 SortList()	22
5.1.3.14 UnlinkNodeByValue()	23
5.1.3.15 WalkToNode()	24
5.2 LinkedList.h	24
5.3 Include/Queue.h File Reference	25
5.3.1 Detailed Description	27
5.3.2 Typedef Documentation	27
5.3.2.1 QUEUE	27
5.3.3 Function Documentation	27
5.3.3.1 Dequeue()	27
5.3.3.2 DestroyQueue()	28
5.3.3.3 Enqueue()	28
5.3.3.4 InitQueue()	29
5.3.3.5 QueueIsEmpty()	30
5.4 Queue.h	30
5.5 Include/Stack.h File Reference	31
5.5.1 Detailed Description	32
5.5.2 Typedef Documentation	32
5.5.2.1 STACK	32
5.5.3 Function Documentation	32
5.5.3.1 DestroyStack()	32
5.5.3.2 InitStack()	33
5.5.3.3 Pop()	33
5.5.3.4 Push()	34
5.5.3.5 StackIsEmpty()	35
5.5.3.6 UpdateCount()	35
5.6 Stack.h	35
5.7 TestRun.h	36
5.8 Include/Utility.h File Reference	36
5.8.1 Detailed Description	37
5.8.2 Function Documentation	37
5.8.2.1 compare_char()	37
5.8.2.2 compare_double()	38
5.8.2.3 compare_float()	38
5.8.2.4 compare_int32_t()	39
5.8.2.5 compare_int64_t()	39
5.8.2.6 compare_long_double()	40
5.8.2.7 compare_string()	40
5.8.2.8 compareIntArray()	40
5.8.2.9 randomInt()	41

5.8.2.10 TestList()	41
5.8.2.11 TestQueue()	41
5.8.2.12 TestStack()	42
5.8.2.13 ToArray()	42
5.9 Utility.h	43
5.10 Src/LinkedList/LinkedList.c File Reference	43
5.10.1 Detailed Description	44
5.10.2 Function Documentation	44
5.10.2.1 Add()	44
5.10.2.2 DestroyList()	45
5.10.2.3 DumpList()	45
5.10.2.4 findMid()	46
5.10.2.5 Get()	46
5.10.2.6 IndexOf()	47
5.10.2.7 InitList()	48
5.10.2.8 InsertNodeAfterTarget()	49
5.10.2.9 InsertNodeBeforeTarget()	49
5.10.2.10 MergeSort()	50
5.10.2.11 RemoveByIndex()	51
5.10.2.12 Sort()	52
5.10.2.13 SortList()	53
5.10.2.14 UnlinkNodeByValue()	54
5.10.2.15 WalkToNode()	55
5.11 Src/Queue/Queue.c File Reference	55
5.11.1 Detailed Description	56
5.11.2 Function Documentation	56
5.11.2.1 Dequeue()	56
5.11.2.2 DestroyQueue()	57
5.11.2.3 Enqueue()	58
5.11.2.4 InitQueue()	58
5.11.2.5 QueueIsEmpty()	59
5.12 Src/Stack/Stack.c File Reference	59
5.12.1 Detailed Description	60
5.12.2 Function Documentation	60
5.12.2.1 DestroyStack()	60
5.12.2.2 InitStack()	61
5.12.2.3 Pop()	61
5.12.2.4 Push()	62
5.12.2.5 StackIsEmpty()	62
5.13 Test/LinkedListTest.c File Reference	64
5.13.1 Detailed Description	65
5.14 Test/QueueTest.c File Reference	65

---

5.14.1 Detailed Description . . . . .	66
5.15 Test/StackTest.c File Reference . . . . .	67
5.15.1 Detailed Description . . . . .	68
5.16 Test/TestRun.c File Reference . . . . .	68
5.16.1 Detailed Description . . . . .	69
5.16.2 Function Documentation . . . . .	69
5.16.2.1 TestAll() . . . . .	69
5.17 Util/Utility.c File Reference . . . . .	69
5.17.1 Detailed Description . . . . .	70
5.17.2 Function Documentation . . . . .	70
5.17.2.1 compare_char() . . . . .	70
5.17.2.2 compare_double() . . . . .	71
5.17.2.3 compare_float() . . . . .	71
5.17.2.4 compare_int32_t() . . . . .	71
5.17.2.5 compare_int64_t() . . . . .	72
5.17.2.6 compare_long_double() . . . . .	72
5.17.2.7 compare_string() . . . . .	73
5.17.2.8 compareIntArray() . . . . .	73
5.17.2.9 randomInt() . . . . .	73
5.17.2.10 TestList() . . . . .	74
5.17.2.11 TestQueue() . . . . .	74
5.17.2.12 TestStack() . . . . .	75
5.17.2.13 ToArray() . . . . .	75
<b>Index</b>	<b>77</b>

# Chapter 1

## Data Structure Library

This is a small data structure library written in C. A data structure is a storage that is used to store and organize data. It is a way of arranging data on a computer so that it can be accessed and updated efficiently. [1](#) The library includes a doubly linked-list, [Queue](#), and [Stack](#) along with test cases for each.

### 1.1 Linked List

A linked-list is a generic data structure that grows and shrinks dynamically. It usually is implemented with a series of individually allocated nodes that point from one item to the next, in a specified order. Nodes can be added and inserted arbitrarily in  $O(1)$  time if a link to the insertion/deletion point is known ahead of time. Searching a linked-list is an  $O(n)$  operation.

There are different types of linked-lists: doubly-linked, circular, and more. We implemented a doubly-linked list that tracks pointers for the head and tail of the list, which point to the first and last node, respectively.

### 1.2 Stack

A stack is a Last In First Out (LIFO) data structure that uses the Push and Pop operations to add and remove data. It is often implemented on top of arrays and linked lists. We use our Linked List Library to implement our [Stack](#).

### 1.3 Queue

The queue data structure is similar to, but opposite of, a stack. It operates in a First In First Out (FIFO) order and uses Enqueue and Dequeue operations to add and remove data. We use our Linked List Library to implement our [Queue](#).

### 1.4 What Data Type?

This library is done with a void pointer data type to make the structures generic. The reason we use a void pointer is because C does not have a generic data type. By using the void pointer, this pointer can point to any type of data.

### 1.4.1 Documentation

A [PDF](#) with all the documentation for our library. The PDF was generated using Doxygen with our header file comments.

### 1.4.2 Installation

Below is one method of installation using Cmake.

1. Download [source code](#)

2. Add into project directory.

3. Inside your 'CMakeLists.txt' add

```
//This sets a variable named SOURCE_FILES to the location of our source files. If your file directory
is different then just adjust to fit your machine.
set(SOURCE_FILES Src/LinkedList/LinkedList.c Util/Utility.c Src/Queue/Queue.c Src/Stack/Stack.c
Test/TestRun.c Test/LinkedListTest.c Test/StackTest.c Test/StackTest.c)

add_library(DataStructures ${SOURCE_FILES}) //creates the library

add_executable({Your name of Executable} {Your source files}) //creates the executable

target_link_libraries({Your name of Executable} PRIVATE DataStructures) // this tells CMake to link
the executable and library
```

4. Run your exe.

### 1.4.3 Authors

- [@tensign1444](#)
- [@michaelVaquilar](#)
- [@Masa-dotcom](#)

### 1.4.4 Feedback

If you have any feedback, please create an [Issue](#).



## Chapter 2

# Data Structure Index

### 2.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">LList</a>	7
<a href="#">Node</a>	8
<a href="#">Queue</a>	8
<a href="#">Stack</a>	9



## Chapter 3

# File Index

### 3.1 File List

Here is a list of all documented files with brief descriptions:

Include/ <a href="#">LinkedList.h</a> . . . . .	11
Include/ <a href="#">Queue.h</a> . . . . .	25
Include/ <a href="#">Stack.h</a> . . . . .	31
Include/ <a href="#">TestRun.h</a> . . . . .	36
Include/ <a href="#">Utility.h</a> . . . . .	36
Src/LinkedList/ <a href="#">LinkedList.c</a> . . . . .	43
Src/Queue/ <a href="#">Queue.c</a> . . . . .	55
Src/Stack/ <a href="#">Stack.c</a> . . . . .	59
Test/ <a href="#">LinkedListTest.c</a> . . . . .	64
Test/ <a href="#">QueueTest.c</a> . . . . .	65
Test/ <a href="#">StackTest.c</a> . . . . .	67
Test/ <a href="#">TestRun.c</a> . . . . .	68
Util/ <a href="#">Utility.c</a> . . . . .	69



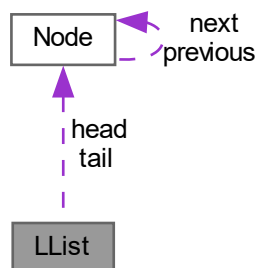
## Chapter 4

# Data Structure Documentation

### 4.1 LList Struct Reference

```
#include <LinkedList.h>
```

Collaboration diagram for LList:



#### Data Fields

- `NODE * head`
- `NODE * tail`
- `int count`
- `compare CompareTo`

#### 4.1.1 Detailed Description

List structure to hold our head, tail, count and compareTo function.

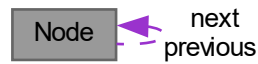
The documentation for this struct was generated from the following file:

- Include/[LinkedList.h](#)

## 4.2 Node Struct Reference

```
#include <LinkedList.h>
```

Collaboration diagram for Node:



### Data Fields

- void \*\* **value**
- struct [Node](#) \* **next**
- struct [Node](#) \* **previous**

#### 4.2.1 Detailed Description

[Node](#) structure to hold the value along with the next and previous nodes.

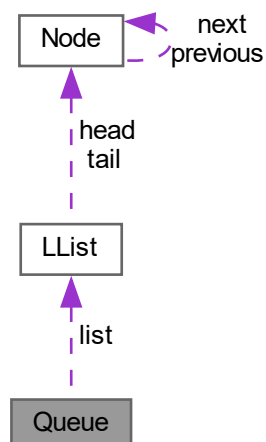
The documentation for this struct was generated from the following file:

- Include/[LinkedList.h](#)

## 4.3 Queue Struct Reference

```
#include <Queue.h>
```

Collaboration diagram for Queue:



## Data Fields

- int **Count**
- **LIST** \* list

### 4.3.1 Detailed Description

**Stack** structure that holds our **Queue** which is an implementation of our linkedlist.

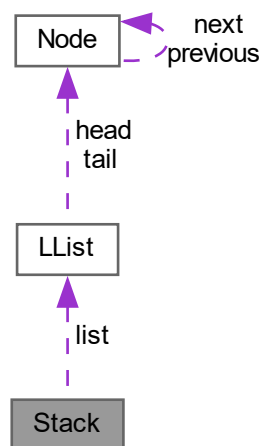
The documentation for this struct was generated from the following file:

- Include/**Queue.h**

## 4.4 Stack Struct Reference

```
#include <Stack.h>
```

Collaboration diagram for Stack:



## Data Fields

- int **Count**
- **LIST** \* list

### 4.4.1 Detailed Description

**Stack** structure that holds ourstack which is an implementation of our linkedlist.

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

- Include/**Stack.h**





## Chapter 5

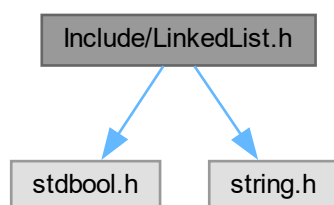
# File Documentation

### 5.1 Include/LinkedList.h File Reference

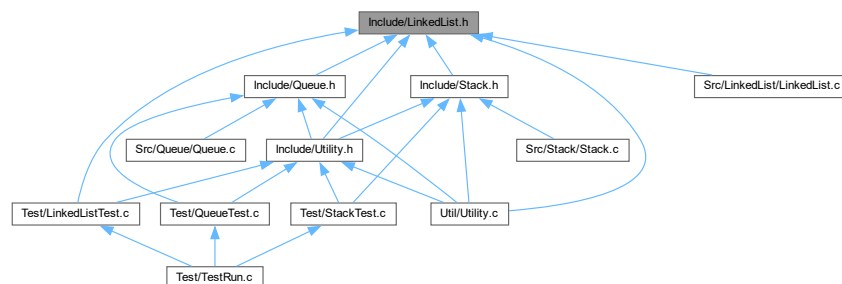
```
#include <stdbool.h>
```

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

Include dependency graph for LinkedList.h:



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



## Data Structures

- struct [Node](#)
- struct [LList](#)

## Typedefs

- typedef int(\* [compare](#)) (const void \*, const void \*)
- typedef struct [Node](#) [NODE](#)
- typedef struct [LList](#) [LIST](#)

## Functions

- [LIST](#) \* [InitList](#) ([compare](#) Compare)
- void [Add](#) ([LIST](#) \*list, void \*value)
- void \* [Get](#) ([LIST](#) \*list, int index)
- void [DestroyList](#) ([LIST](#) \*list)
- void [DumpList](#) ([LIST](#) \*list)
- int [IndexOf](#) ([LIST](#) \*list, void \*value)
- void [InsertNodeBeforeTarget](#) ([LIST](#) \*list, int index, void \*newValue)
- void [InsertNodeAfterTarget](#) ([LIST](#) \*list, int index, void \*newValue)
- bool [UnlinkNodeByValue](#) ([LIST](#) \*list, void \*value)
- void \* [RemoveByIndex](#) ([LIST](#) \*list, int index)
- [NODE](#) \* [WalkToNode](#) ([NODE](#) \*temp, int location)
- [NODE](#) \* [findMid](#) ([NODE](#) \*start)
- [NODE](#) \* [Sort](#) ([LIST](#) \*list, [NODE](#) \*leftCursor, [NODE](#) \*rightCursor)
- [NODE](#) \* [MergeSort](#) ([LIST](#) \*list, [NODE](#) \*start)
- void [SortList](#) ([LIST](#) \*list)

### 5.1.1 Detailed Description

#### Author

Tanner Ensign, Michael Vaquilar, Masaya Takahashi

#### Date

1/31/2023

### 5.1.2 Typedef Documentation

### 5.1.2.1 compare

```
typedef int(* compare) (const void *, const void *)
```

function that compares two elements.

#### Returns

1 when the first element is greater than the second, -1 when the first element is less than the second, 0 when both elements are equal.

### 5.1.2.2 LIST

```
typedef struct LList LIST
```

List structure to hold our head, tail, count and compareTo function.

### 5.1.2.3 NODE

```
typedef struct Node NODE
```

[Node](#) structure to hold the value along with the next and previous nodes.

## 5.1.3 Function Documentation

### 5.1.3.1 Add()

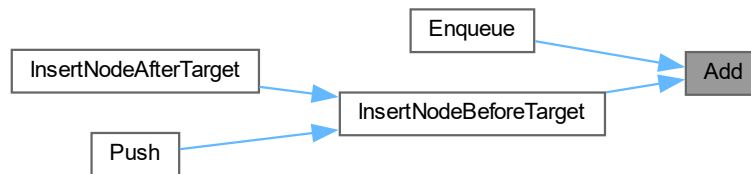
```
void Add (
    LIST * list,
    void * value )
```

Adds a value to the linked list.

#### Parameters

<i>list</i>	to Add too.
<i>value</i>	to be added.

Here is the caller graph for this function:



### 5.1.3.2 DestroyList()

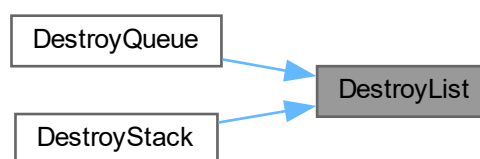
```
void DestroyList (
    LIST * list )
```

Destroys the list aka get's rid of the memory location by freeing the allocated memory.

#### Parameters

<i>list</i>	to destroy.
-------------	-------------

Here is the caller graph for this function:



### 5.1.3.3 DumpList()

```
void DumpList (
    LIST * list )
```

Prints list to console line by line

## Parameters

<i>list</i>	to write to console
-------------	---------------------

### 5.1.3.4 findMid()

```
NODE * findMid (
    NODE * start )
```

Finds the middle of the list.

## Parameters

<i>start</i>	Node
--------------	------

## Returns

the middle node

Here is the caller graph for this function:



### 5.1.3.5 Get()

```
void * Get (
    LIST * list,
    int index )
```

Gets the value at the specific index.

## Parameters

<i>list</i>	to search.
<i>index</i>	to get value at.

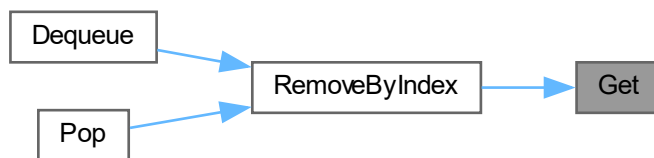
**Returns**

the value at the index.

Here is the call graph for this function:



Here is the caller graph for this function:

**5.1.3.6 IndexOf()**

```
int IndexOf (  
    LIST * list,  
    void * value )
```

Finds the index of a specific value.

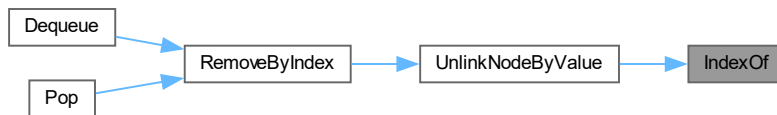
**Parameters**

<i>list</i>	to search.
<i>value</i>	to look for.

**Returns**

int, the index of the value.

Here is the caller graph for this function:

**5.1.3.7 InitList()**

```
LIST * InitList (
    compare Compare )
```

Initializes our linked list so we have a memory location for it. Using calloc so that the memory is already set to 0 instead of empty.

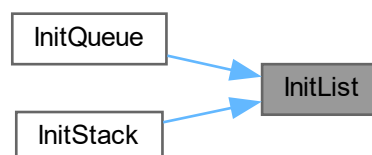
**Parameters**

<i>Compare</i>	compareable function for the list.
----------------	------------------------------------

**Returns**

memory location of the list created, NULL if it failed to allocate memory.

Here is the caller graph for this function:



### 5.1.3.8 InsertNodeAfterTarget()

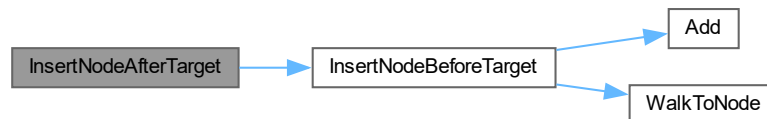
```
void InsertNodeAfterTarget (
    LIST * list,
    int index,
    void * newValue )
```

Inserts the new [Node](#) (aka value) after the specified index.

#### Parameters

<i>list</i>	to insert <a href="#">Node</a> in.
<i>index</i>	to insert new value at.
<i>newValue</i>	the new value to insert.

Here is the call graph for this function:



### 5.1.3.9 InsertNodeBeforeTarget()

```
void InsertNodeBeforeTarget (
    LIST * list,
    int index,
    void * newValue )
```

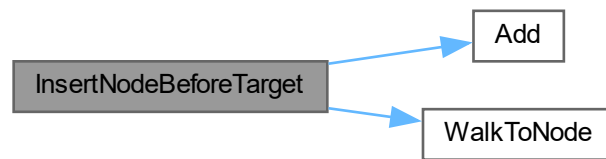
Inserts a node (aka value) before a specific index.

#### Parameters

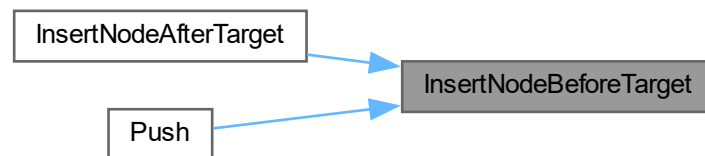
<i>list</i>	to insert <a href="#">Node</a> in.
<i>index</i>	to insert new value at.
<i>newValue</i>	new value to be inserted.



Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.1.3.10 MergeSort()

```

NODE * MergeSort (
    LIST * list,
    NODE * start )
  
```

Uses merge sort to sort the linkedlist in ascending order.

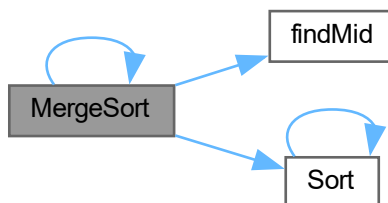
##### Parameters

<i>list</i>	to sort.
<i>start</i>	node

**Returns**

the new sorted node or list

Here is the call graph for this function:



Here is the caller graph for this function:

**5.1.3.11 RemoveByIndex()**

```
void * RemoveByIndex (
    LIST * list,
    int index )
```

Removes a specific index.

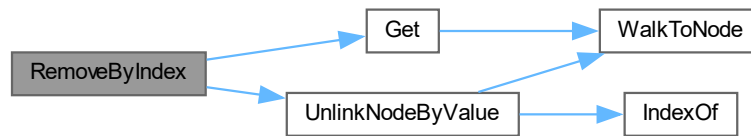
**Parameters**

<i>list</i>	to remove value from.
<i>index</i>	to remove

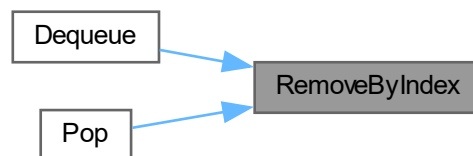
**Returns**

the data that was stored at the node that was removed.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.1.3.12 Sort()

```

NODE * Sort (
    LIST * list,
    NODE * leftCursor,
    NODE * rightCursor )
  
```

Sorts the two nodes given.

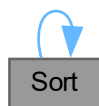
##### Parameters

<i>list</i>	that the values are from.
<i>leftCursor</i>	the left data value
<i>rightCursor</i>	the right data value

**Returns**

new sorted node.

Here is the call graph for this function:



Here is the caller graph for this function:

**5.1.3.13 SortList()**

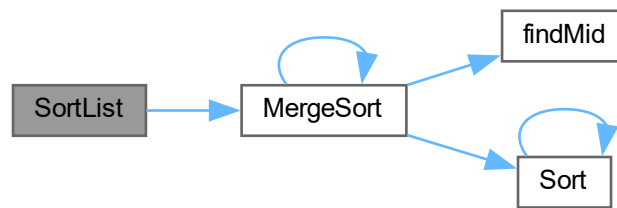
```
void SortList (  
    LIST * list )
```

Calls MergeSort. Simple function for user to call

**Parameters**

<i>list</i>	to sort.
-------------	----------

Here is the call graph for this function:



#### 5.1.3.14 UnlinkNodeByValue()

```
bool UnlinkNodeByValue (  
    LIST * list,  
    void * value )
```

Removes a node by the specified value.

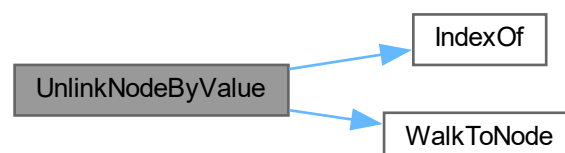
##### Parameters

<i>list</i>	to unlink <a href="#">Node</a> from.
<i>value</i>	in the node we want to remove.

##### Returns

true if the node was removed, false otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.1.3.15 WalkToNode()

```

NODE * WalkToNode (
    NODE * temp,
    int location )
  
```

Walks to an index inside a specific node.

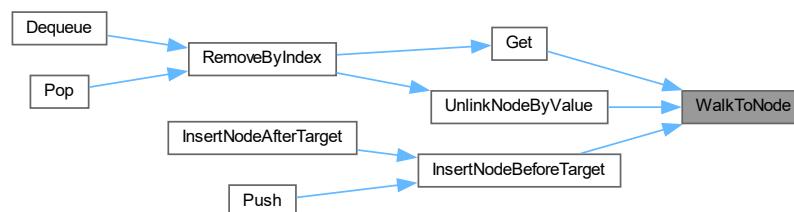
#### Parameters

<i>temp</i>	the node to walk.
<i>location</i>	to walk to.

#### Returns

NODE, the node walked to.

Here is the caller graph for this function:



## 5.2 LinkedList.h

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

```

00001
00009 #ifndef DATASTRUCTURES_LINKEDLIST_H
00010 #define DATASTRUCTURES_LINKEDLIST_H
00011
00012 #include <stdbool.h>
00013 #include <string.h>
00014
00019 typedef int (*compare)(const void *,const void *);
00020
00021
00022
00026 typedef struct Node{
00027     void **value;
00028     struct Node *next;
00029     struct Node *previous;
00030 } NODE;
00031
00035 typedef struct LList{
00036     NODE *head;
00037     NODE *tail;
00038     int count;
00039     compare CompareTo;
00040 } LIST;
00041
00042
00049 LIST* InitList(compare Compare);
00050
00056 void Add(LIST *list, void *value);
00057
00064 void *Get(LIST *list, int index);
00065
00070 void DestroyList(LIST *list);
00071
00076 void DumpList(LIST *list);
00077
00084 int IndexOf(LIST *list,void *value);
00085
00092 void InsertNodeBeforeTarget(LIST *list, int index, void *newValue);
00093
00100 void InsertNodeAfterTarget(LIST *list, int index, void *newValue);
00101
00108 bool UnlinkNodeByValue(LIST *list, void *value);
00109
00116 void *RemoveByIndex(LIST *list, int index);
00117
00124 NODE *WalkToNode(NODE *temp,int location);
00125
00131 NODE *findMid(NODE *start);
00132
00140 NODE *Sort(LIST *list, NODE *leftCursor, NODE *rightCursor);
00141
00148 NODE *MergeSort(LIST *list, NODE *start);
00149
00154 void SortList(LIST *list);
00155
00156 #endif //DATASTRUCTURES_LINKEDLIST_H

```

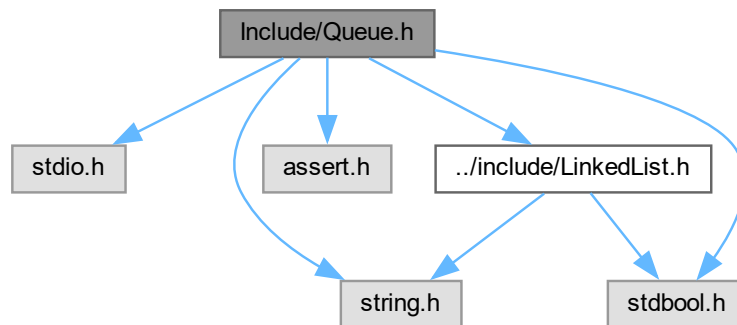
## 5.3 Include/Queue.h File Reference

```

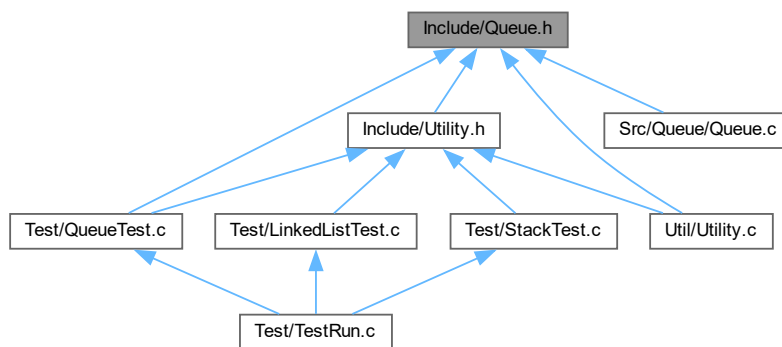
#include <stdio.h>
#include <string.h>
#include <assert.h>
#include <stdbool.h>
#include "../include/LinkedList.h"

```

Include dependency graph for Queue.h:



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



## Data Structures

- struct `Queue`

## Typedefs

- typedef struct `Queue` `QUEUE`

## Functions

- `QUEUE * InitQueue (compare Compare)`
- void `Enqueue (QUEUE *ourQueue, void *item)`
- void \* `Dequeue (QUEUE *ourQueue)`
- bool `QueueIsEmpty (QUEUE *ourQueue)`
- void `DestroyQueue (QUEUE *ourQueue)`



### 5.3.1 Detailed Description

#### Author

Tanner Ensign, Michael Vaquilar, Masaya Takahashi

#### Date

1/31/2023

### 5.3.2 Typedef Documentation

#### 5.3.2.1 QUEUE

```
typedef struct Queue QUEUE
```

[Stack](#) structure that holds our [Queue](#) which is an implementation of our linkedlist.

### 5.3.3 Function Documentation

#### 5.3.3.1 Dequeue()

```
void * Dequeue (  
    QUEUE * ourQueue )
```

Dequeues the item at the front of the list (removes the first item).

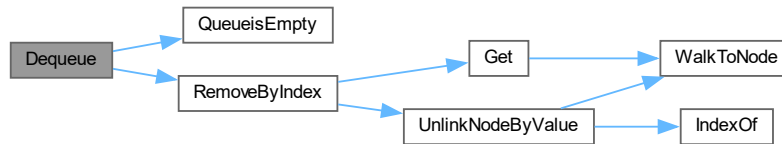
#### Parameters

<i>ourQueue</i>	the queue to dequeue an item from.
-----------------	------------------------------------

**Returns**

the item removed.

Here is the call graph for this function:

**5.3.3.2 DestroyQueue()**

```
void DestroyQueue (
    QUEUE * ourQueue )
```

Destroys the queue, aka freeing the memory

**Parameters**

<code>ourQueue</code>	the queue to destroy.
-----------------------	-----------------------

Here is the call graph for this function:

**5.3.3.3 Enqueue()**

```
void Enqueue (
    QUEUE * ourQueue,
    void * item )
```

Add's an item to the end of the queue.

## Parameters

<i>ourQueue</i>	the queue to enqueue onto.
<i>item</i>	void * item to add.

Here is the call graph for this function:



#### 5.3.3.4 InitQueue()

```
QUEUE * InitQueue (
    compare Compare )
```

Initializes our queue and allocates memory for the queue and list.

## Parameters

<i>Compare</i>	a compare function for the generic data type.
----------------	---

## Returns

pointer to the queue created, null if it couldn't be created.

Here is the call graph for this function:



### 5.3.3.5 QueueisEmpty()

```
bool QueueisEmpty (
    QUEUE * ourQueue )
```

Checks if the queue is empty.

#### Parameters

<i>ourQueue</i>	the queue to check if it is empty.
-----------------	------------------------------------

#### Returns

true if empty, false otherwise.

Here is the caller graph for this function:



## 5.4 Queue.h

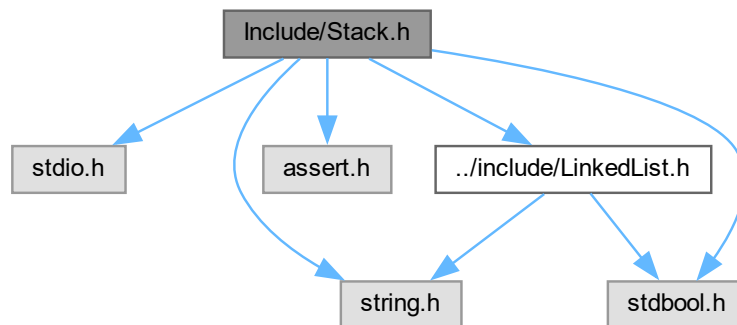
[Go to the documentation of this file.](#)

```
00001
00009 #ifndef DATASTRUCTURES_QUEUE_H
00010 #define DATASTRUCTURES_QUEUE_H
00011
00012 #include <stdio.h>
00013 #include <string.h>
00014 #include <assert.h>
00015 #include <stdbool.h>
00016 #include "../include/LinkedList.h"
00017
00021 typedef struct Queue{
00022     int Count;
00023     LIST *list;
00024 }QUEUE;
00025
00031 QUEUE* InitQueue(compare Compare);
00032
00038 void Enqueue(QUEUE *ourQueue,void *item);
00039
00045 void* Dequeue(QUEUE *ourQueue);
00046
00052 bool QueueisEmpty(QUEUE *ourQueue);
00053
00058 void DestroyQueue(QUEUE *ourQueue);
00059
00060 #endif //DATASTRUCTURES_QUEUE_H
```

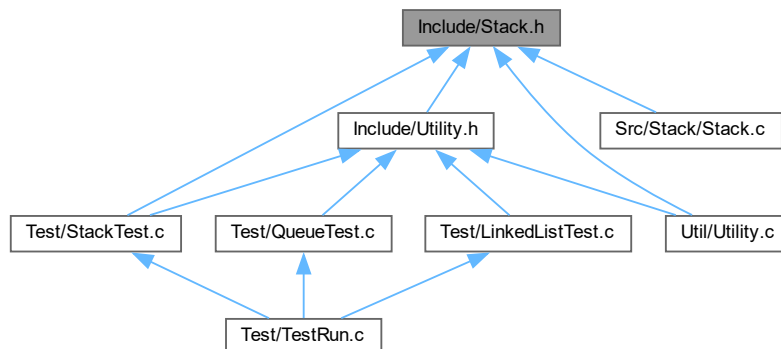
## 5.5 Include/Stack.h File Reference

```
#include <stdio.h>
#include <string.h>
#include <assert.h>
#include <stdbool.h>
#include "../include/LinkedList.h"
```

Include dependency graph for Stack.h:



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



## Data Structures

- struct [Stack](#)

## Typedefs

- typedef struct [Stack](#) [STACK](#)

## Functions

- void [UpdateCount](#) ()
- [STACK](#) \* [InitStack](#) ([compare](#) Compare)
- bool [StackIsEmpty](#) ([STACK](#) \*ourStack)
- void \* [Pop](#) ([STACK](#) \*ourStack)
- void [Push](#) ([STACK](#) \*ourStack, void \*data)
- void [DestroyStack](#) ([STACK](#) \*ourStack)

### 5.5.1 Detailed Description

#### Author

Tanner Ensign, Michael Vaquilar, Masaya Takahashi

#### Date

1/31/2023

### 5.5.2 Typedef Documentation

#### 5.5.2.1 STACK

```
typedef struct Stack STACK
```

[Stack](#) structure that holds ourstack which is an implementation of our linkedlist.

### 5.5.3 Function Documentation

#### 5.5.3.1 DestroyStack()

```
void DestroyStack (  
    STACK * ourStack )
```

Destroys the stack, aka freeing the memory.

#### Parameters

<i>ourStack</i>	the stack to destroy.
-----------------	-----------------------

Here is the call graph for this function:



### 5.5.3.2 InitStack()

```
STACK * InitStack (
    compare Compare )
```

Initializes our stack and allocates memory for the queue and list.

#### Parameters

<i>Compare</i>	a compare function for the generic data type.
----------------	---

#### Returns

pointer to the stack made, NULL if it couldn't be made.

Here is the call graph for this function:



### 5.5.3.3 Pop()

```
void * Pop (
    STACK * ourStack )
```

Pop's an item off the top of the stack.

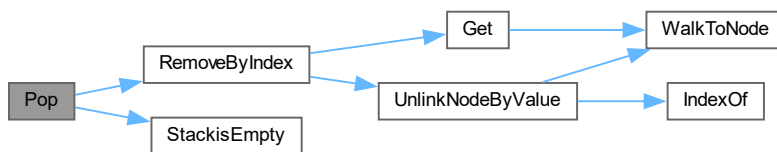
**Parameters**

<i>ourStack</i>	the stack to pop an item off of.
-----------------	----------------------------------

**Returns**

the generic pointer to the item popped off.

Here is the call graph for this function:

**5.5.3.4 Push()**

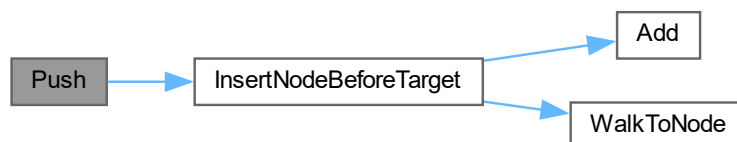
```
void Push (
    STACK * ourStack,
    void * data )
```

Pushes a generic pointer to an item onto the top of the stack.

**Parameters**

<i>ourStack</i>	the stack to push onto.
<i>data</i>	void pointer to the data to pop onto the stack.

Here is the call graph for this function:





### 5.5.3.5 StackIsEmpty()

```
bool StackIsEmpty (
    STACK * ourStack )
```

Checks if the stack is empty.

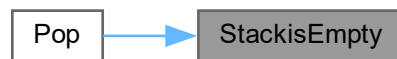
#### Parameters

<i>ourStack</i>	the stack to check if empty.
-----------------	------------------------------

#### Returns

true if empty, false otherwise.

Here is the caller graph for this function:



### 5.5.3.6 UpdateCount()

```
void UpdateCount ( )
```

Updates the count of the [Stack](#).

## 5.6 Stack.h

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

```
00001
00009 #ifndef DATASTRUCTURES_STACK_H
00010 #define DATASTRUCTURES_STACK_H
00011
00012 #include <stdio.h>
00013 #include <string.h>
00014 #include <assert.h>
00015 #include <stdbool.h>
00016 #include "../include/LinkedList.h"
00017
00021 typedef struct Stack{
00022     int Count;
00023     LIST *list;
00024 }STACK;
00025
00029 void UpdateCount();
00030
00036 STACK* InitStack(compare Compare);
```

```

00037
00043 bool StackIsEmpty(STACK *ourStack);
00044
00050 void *Pop(STACK *ourStack);
00051
00057 void Push(STACK *ourStack, void *data);
00058
00063 void DestroyStack(STACK *ourStack);
00064
00065 #endif //DATASTRUCTURES_STACK_H

```

## 5.7 TestRun.h

```

00001
00008 #ifndef DATASTRUCTURES_TESTRUN_H
00009 #define DATASTRUCTURES_TESTRUN_H
00010
00014 void TestAll();
00015
00016 #endif //DATASTRUCTURES_TESTRUN_H

```

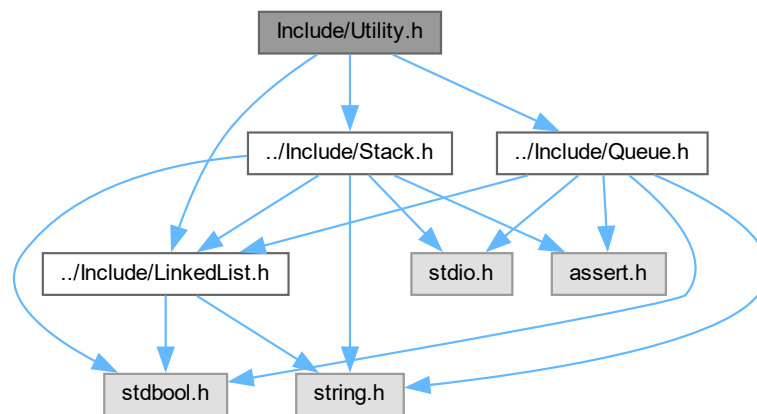
## 5.8 Include/Utility.h File Reference

```

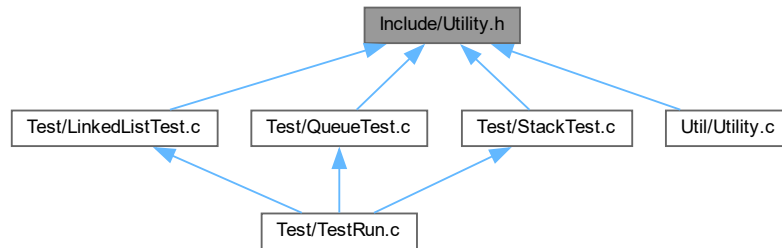
#include "../Include/LinkedList.h"
#include "../Include/Queue.h"
#include "../Include/Stack.h"

```

Include dependency graph for Utility.h:



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



## Functions

- int [compare\\_int32\\_t](#) (const void \*element1, const void \*element2)
- int [compare\\_int64\\_t](#) (const void \*element1, const void \*element2)
- int [compare\\_float](#) (const void \*element1, const void \*element2)
- int [compare\\_double](#) (const void \*element1, const void \*element2)
- int [compare\\_long\\_double](#) (const void \*element1, const void \*element2)
- int [compare\\_char](#) (const void \*element1, const void \*element2)
- int [compare\\_string](#) (const void \*element1, const void \*element2)
- int [compareIntArray](#) (int a[], int b[])
- void [TestList](#) (LIST \*listHolder, void \*expected, void \*actual, const char \*testName, bool isNULL)
- void [TestQueue](#) (QUEUE \*queue, void \*expected, void \*actual, const char \*testName, bool isNULL)
- void [TestStack](#) (STACK \*stack, void \*expected, void \*actual, const char \*testName, bool isNULL)
- int [randomInt](#) ()
- int \* [ToArray](#) (LIST \*listHolder)

### 5.8.1 Detailed Description

#### Author

Tanner Ensign, Michael Vaquilar, Masaya Takahashi

#### Date

1/31/2023

### 5.8.2 Function Documentation

#### 5.8.2.1 [compare\\_char\(\)](#)

```

int compare_char (
    const void * element1,
    const void * element2 )

```

Compares two char's to each-other.

**Parameters**

<i>element1</i>	void pointer to the first char.
<i>element2</i>	void pointer to the second char.

**Returns**

int, 1 when the first element is greater than the second, -1 when the first element is less than the second, 0 when both elements are equal.

**5.8.2.2 compare\_double()**

```
int compare_double (
    const void * element1,
    const void * element2 )
```

Compares two double's to each-other.

**Parameters**

<i>element1</i>	void pointer to the first double.
<i>element2</i>	void pointer to the second double.

**Returns**

int, 1 when the first element is greater than the second, -1 when the first element is less than the second, 0 when both elements are equal.

**5.8.2.3 compare\_float()**

```
int compare_float (
    const void * element1,
    const void * element2 )
```

Compares two floats to each-other.

**Parameters**

<i>element1</i>	void pointer to the first float.
<i>element2</i>	void pointer to the second float.

**Returns**

int, 1 when the first element is greater than the second, -1 when the first element is less than the second, 0 when both elements are equal.

**5.8.2.4 compare\_int32\_t()**

```
int compare_int32_t (
    const void * element1,
    const void * element2 )
```

Compares two 32 bit integers to each-other.

**Parameters**

<i>element1</i>	void pointer to the first integer.
<i>element2</i>	void pointer to the second integer.

**Returns**

int, 1 when the first element is greater than the second, -1 when the first element is less than the second, 0 when both elements are equal.

**5.8.2.5 compare\_int64\_t()**

```
int compare_int64_t (
    const void * element1,
    const void * element2 )
```

Compares two 64 bit integers to each-other.

**Parameters**

<i>element1</i>	void pointer to the first integer.
<i>element2</i>	void pointer to the second integer.

**Returns**

int, 1 when the first element is greater than the second, -1 when the first element is less than the second, 0 when both elements are equal.

### 5.8.2.6 compare\_long\_double()

```
int compare_long_double (
    const void * element1,
    const void * element2 )
```

Compares two long doubles to each-other.

#### Parameters

<i>element1</i>	void pointer to the first long double.
<i>element2</i>	void pointer to the second long double.

#### Returns

int, 1 when the first element is greater than the second, -1 when the first element is less than the second, 0 when both elements are equal.

### 5.8.2.7 compare\_string()

```
int compare_string (
    const void * element1,
    const void * element2 )
```

Compares two strings to each-other.

#### Parameters

<i>element1</i>	void pointer to the first string.
<i>element2</i>	void pointer to the second string.

#### Returns

int, 1 when the first element is greater than the second, -1 when the first element is less than the second, 0 when both elements are equal.

### 5.8.2.8 compareIntArray()

```
int compareIntArray (
    int a[],
    int b[] )
```

Compares to int arrays to see if they are similar.

## Parameters

<i>a</i>	first int array.
<i>b</i>	second int array.

## Returns

0 if not, 1 if equal.

**5.8.2.9 randomInt()**

```
int randomInt ( )
```

Generates a random 32 bit int.

## Returns

randomly generated int.

**5.8.2.10 TestList()**

```
void TestList (
    LIST * listHolder,
    void * expected,
    void * actual,
    const char * testName,
    bool isNULL )
```

Small method for a list unit test, checks if two objects are equal, if so they passed.

## Parameters

<i>listHolder</i>	linkedList to test
<i>expected</i>	output
<i>actual</i>	output
<i>testName</i>	name of the test
<i>isNULL</i>	check if actual and expected both should be NULL

**5.8.2.11 TestQueue()**

```
void TestQueue (
    QUEUE * queue,
```

```
void * expected,
void * actual,
const char * testName,
bool isNULL )
```

Small method for a list [Queue](#) test, checks if two objects are equal, if so they passed.

#### Parameters

<i>queue</i>	to test
<i>expected</i>	output
<i>actual</i>	output
<i>testName</i>	name of the test
<i>isNULL</i>	check if actual and expected both should be NULL

#### 5.8.2.12 TestStack()

```
void TestStack (
    STACK * stack,
    void * expected,
    void * actual,
    const char * testName,
    bool isNULL )
```

Small method for a [Stack](#) unit test, checks if two objects are equal, if so they passed.

#### Parameters

<i>stack</i>	to test
<i>expected</i>	output
<i>actual</i>	output
<i>testName</i>	name of the test
<i>isNULL</i>	check if actual and expected both should be NULL

#### 5.8.2.13 ToArray()

```
int * ToArray (
    LIST * listHolder )
```

Converts a int LinkedList into an int array

#### Parameters

<i>listHolder</i>	the list to convert into an array
-------------------	-----------------------------------



## Returns

int array

## 5.9 Utility.h

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

```

00001
00009 #ifndef DATASTRUCTURES_UTILITY_H
00010 #define DATASTRUCTURES_UTILITY_H
00011
00012 #include "../Include/LinkedList.h"
00013 #include "../Include/Queue.h"
00014 #include "../Include/Stack.h"
00015
00022 int compare_int32_t(const void *element1, const void *element2);
00023
00030 int compare_int64_t(const void *element1, const void *element2);
00031
00038 int compare_float(const void *element1, const void *element2);
00039
00046 int compare_double(const void *element1, const void *element2);
00047
00054 int compare_long_double(const void *element1, const void *element2);
00055
00062 int compare_char(const void *element1, const void *element2);
00063
00070 int compare_string(const void *element1, const void *element2);
00071
00078 int compareIntArray(int a[], int b[]);
00079
00088 void TestList(LIST *listHolder, void *expected, void *actual, const char* testName, bool isNULL);
00089
00098 void TestQueue(Queue *queue, void *expected, void *actual, const char* testName, bool isNULL);
00099
00108 void TestStack(STACK *stack, void *expected, void *actual, const char* testName, bool isNULL);
00109
00114 int randomInt();
00115
00121 int * ToArray(LIST *listHolder);
00122
00123 #endif //DATASTRUCTURES_UTILITY_H

```

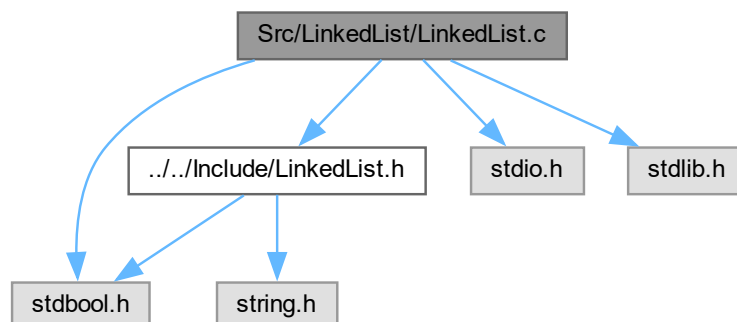
## 5.10 Src/LinkedList/LinkedList.c File Reference

```

#include "../Include/LinkedList.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

```

Include dependency graph for LinkedList.c:



## Functions

- `LIST * InitList (compare Compare)`
- `NODE * GetHead (NODE *temp)`
- `NODE * GetTail (NODE *temp)`
- `void Add (LIST *list, void *value)`
- `void * Get (LIST *list, int index)`
- `void DestroyList (LIST *list)`
- `void DumpList (LIST *list)`
- `int IndexOf (LIST *list, void *value)`
- `void InsertNodeBeforeTarget (LIST *list, int index, void *newValue)`
- `void InsertNodeAfterTarget (LIST *list, int index, void *newValue)`
- `bool UnlinkNodeByValue (LIST *list, void *value)`
- `void * RemoveByIndex (LIST *list, int index)`
- `NODE * WalkToNode (NODE *temp, int location)`
- `NODE * findMid (NODE *start)`
- `NODE * Sort (LIST *list, NODE *leftCursor, NODE *rightCursor)`
- `NODE * MergeSort (LIST *list, NODE *start)`
- `void SortList (LIST *list)`

### 5.10.1 Detailed Description

#### Author

Tanner Ensign, Michael Vaquilar, Masaya Takahashi

#### Date

1/31/2023

### 5.10.2 Function Documentation

#### 5.10.2.1 Add()

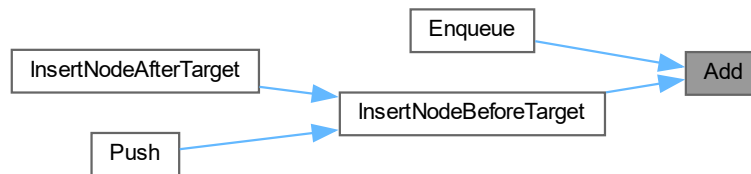
```
void Add (
    LIST * list,
    void * value )
```

Adds a value to the linked list.

#### Parameters

<i>list</i>	to Add too.
<i>value</i>	to be added.

Here is the caller graph for this function:



### 5.10.2.2 DestroyList()

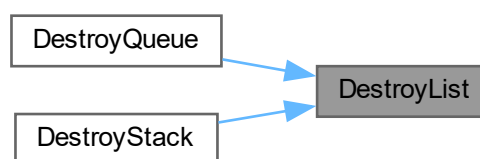
```
void DestroyList (  
    LIST * list )
```

Destroys the list aka get's rid of the memory location by freeing the allocated memory.

#### Parameters

<i>list</i>	to destroy.
-------------	-------------

Here is the caller graph for this function:



### 5.10.2.3 DumpList()

```
void DumpList (  
    LIST * list )
```

Prints list to console line by line

**Parameters**

<i>list</i>	to write to console
-------------	---------------------

**5.10.2.4 findMid()**

```
NODE * findMid (
    NODE * start )
```

Finds the middle of the list.

**Parameters**

<i>start</i>	Node
--------------	------

**Returns**

the middle node

Here is the caller graph for this function:

**5.10.2.5 Get()**

```
void * Get (
    LIST * list,
    int index )
```

Gets the value at the specific index.

**Parameters**

<i>list</i>	to search.
<i>index</i>	to get value at.

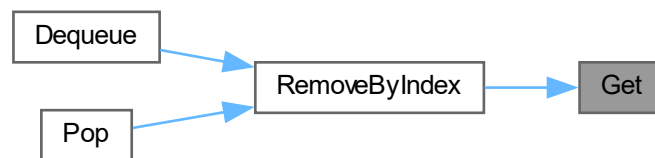
**Returns**

the value at the index.

Here is the call graph for this function:



Here is the caller graph for this function:

**5.10.2.6 IndexOf()**

```
int IndexOf (
    LIST * list,
    void * value )
```

Finds the index of a specific value.

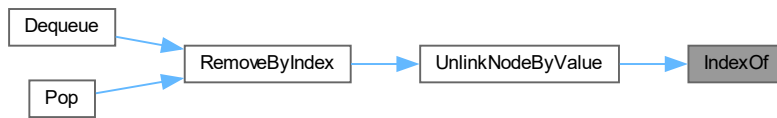
**Parameters**

<i>list</i>	to search.
<i>value</i>	to look for.

**Returns**

int, the index of the value.

Here is the caller graph for this function:

**5.10.2.7 InitList()**

```
LIST * InitList (
    compare Compare )
```

Initializes our linked list so we have a memory location for it. Using calloc so that the memory is already set to 0 instead of empty.

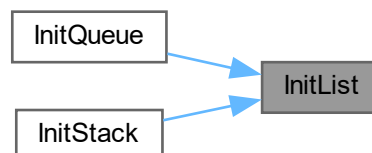
**Parameters**

<i>Compare</i>	compareable function for the list.
----------------	------------------------------------

**Returns**

memory location of the list created, NULL if it failed to allocate memory.

Here is the caller graph for this function:



### 5.10.2.8 InsertNodeAfterTarget()

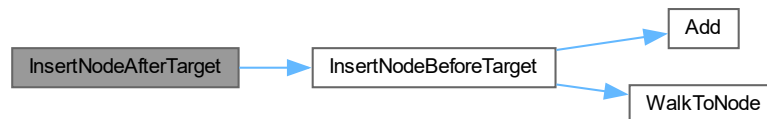
```
void InsertNodeAfterTarget (
    LIST * list,
    int index,
    void * newValue )
```

Inserts the new [Node](#) (aka value) after the specified index.

#### Parameters

<i>list</i>	to insert <a href="#">Node</a> in.
<i>index</i>	to insert new value at.
<i>newValue</i>	the new value to insert.

Here is the call graph for this function:



### 5.10.2.9 InsertNodeBeforeTarget()

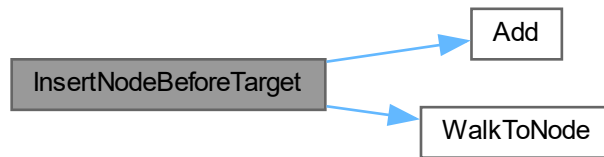
```
void InsertNodeBeforeTarget (
    LIST * list,
    int index,
    void * newValue )
```

Inserts a node (aka value) before a specific index.

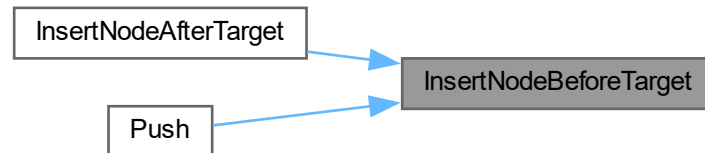
#### Parameters

<i>list</i>	to insert <a href="#">Node</a> in.
<i>index</i>	to insert new value at.
<i>newValue</i>	new value to be inserted.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.10.2.10 MergeSort()

```

NODE * MergeSort (
    LIST * list,
    NODE * start )
  
```

Uses merge sort to sort the linkedlist in ascending order.

##### Parameters

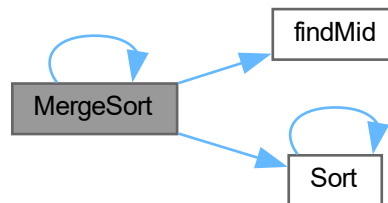
<i>list</i>	to sort.
<i>start</i>	node



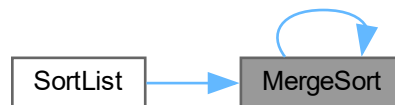
**Returns**

the new sorted node or list

Here is the call graph for this function:



Here is the caller graph for this function:

**5.10.2.11 RemoveByIndex()**

```
void * RemoveByIndex (
    LIST * list,
    int index )
```

Removes a specific index.

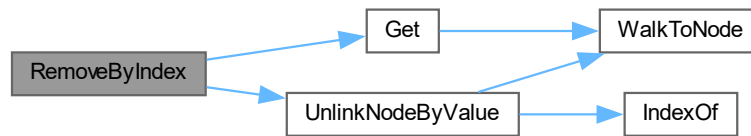
**Parameters**

<i>list</i>	to remove value from.
<i>index</i>	to remove

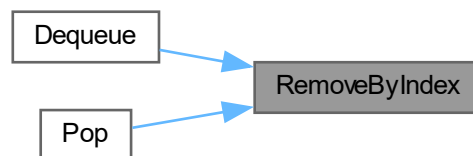
**Returns**

the data that was stored at the node that was removed.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 5.10.2.12 Sort()

```

NODE * Sort (
    LIST * list,
    NODE * leftCursor,
    NODE * rightCursor )
  
```

Sorts the two nodes given.

##### Parameters

<i>list</i>	that the values are from.
<i>leftCursor</i>	the left data value
<i>rightCursor</i>	the right data value

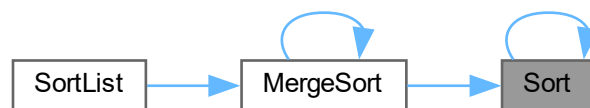
**Returns**

new sorted node.

Here is the call graph for this function:



Here is the caller graph for this function:

**5.10.2.13 SortList()**

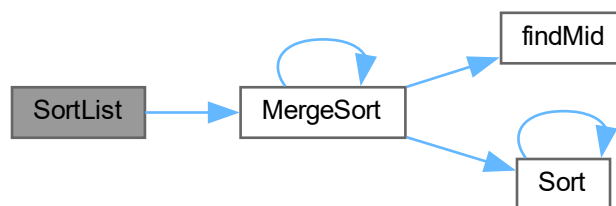
```
void SortList (  
    LIST * list )
```

Calls MergeSort. Simple function for user to call

**Parameters**

<i>list</i>	to sort.
-------------	----------

Here is the call graph for this function:



#### 5.10.2.14 UnlinkNodeByValue()

```
bool UnlinkNodeByValue (
    LIST * list,
    void * value )
```

Removes a node by the specified value.

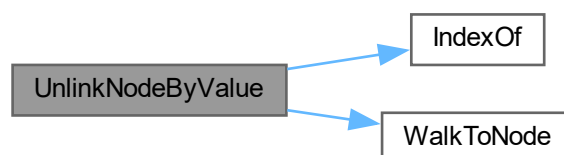
##### Parameters

<i>list</i>	to unlink <a href="#">Node</a> from.
<i>value</i>	in the node we want to remove.

##### Returns

true if the node was removed, false otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:



### 5.10.2.15 WalkToNode()

```

NODE * WalkToNode (
    NODE * temp,
    int location )
  
```

Walks to an index inside a specific node.

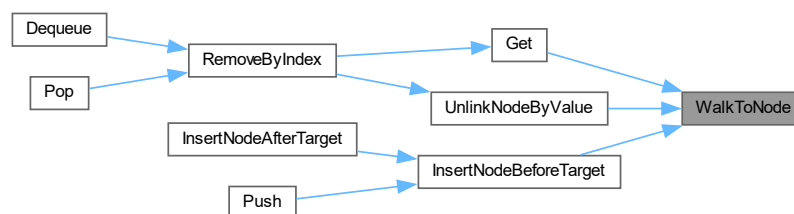
#### Parameters

<i>temp</i>	the node to walk.
<i>location</i>	to walk to.

#### Returns

NODE, the node walked to.

Here is the caller graph for this function:

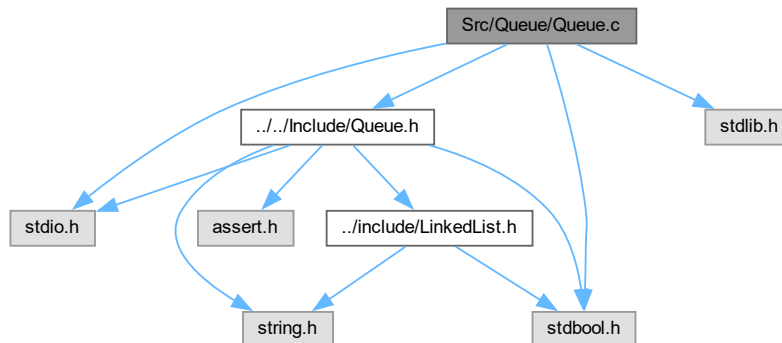


## 5.11 Src/Queue/Queue.c File Reference

```

#include "../Include/Queue.h"
#include <stdio.h>
  
```

```
#include <stdlib.h>
#include <stdbool.h>
Include dependency graph for Queue.c:
```



## Functions

- `QUEUE * InitQueue (compare Compare)`
- `void Enqueue (QUEUE *ourQueue, void *item)`
- `void * Dequeue (QUEUE *ourQueue)`
- `bool QueueIsEmpty (QUEUE *ourQueue)`
- `void DestroyQueue (QUEUE *ourQueue)`

### 5.11.1 Detailed Description

#### Author

Tanner Ensign, Michael Vaquilar, Masaya Takahashi

#### Date

1/31/2023

### 5.11.2 Function Documentation

#### 5.11.2.1 Dequeue()

```
void * Dequeue (
    QUEUE * ourQueue )
```

Dequeues the item at the front of the list (removes the first item).

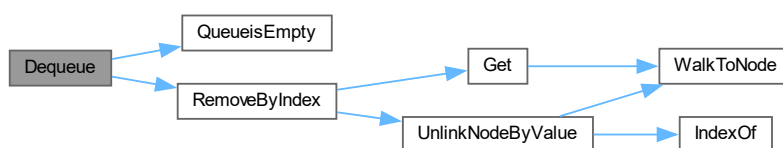
## Parameters

<i>ourQueue</i>	the queue to dequeue an item from.
-----------------	------------------------------------

## Returns

the item removed.

Here is the call graph for this function:



### 5.11.2.2 DestroyQueue()

```
void DestroyQueue (  
    QUEUE * ourQueue )
```

Destroys the queue, aka freeing the memory

## Parameters

<i>ourQueue</i>	the queue to destroy.
-----------------	-----------------------

Here is the call graph for this function:



### 5.11.2.3 Enqueue()

```
void Enqueue (
    QUEUE * ourQueue,
    void * item )
```

Add's an item to the end of the queue.

#### Parameters

<i>ourQueue</i>	the queue to enqueue onto.
<i>item</i>	void * item to add.

Here is the call graph for this function:



### 5.11.2.4 InitQueue()

```
QUEUE * InitQueue (
    compare Compare )
```

Initializes our queue and allocates memory for the queue and list.

#### Parameters

<i>Compare</i>	a compare function for the generic data type.
----------------	---

#### Returns

pointer to the queue created, null if it couldn't be created.

Here is the call graph for this function:





### 5.11.2.5 QueueIsEmpty()

```
bool QueueIsEmpty (
    QUEUE * ourQueue )
```

Checks if the queue is empty.

#### Parameters

<i>ourQueue</i>	the queue to check if it is empty.
-----------------	------------------------------------

#### Returns

true if empty, false otherwise.

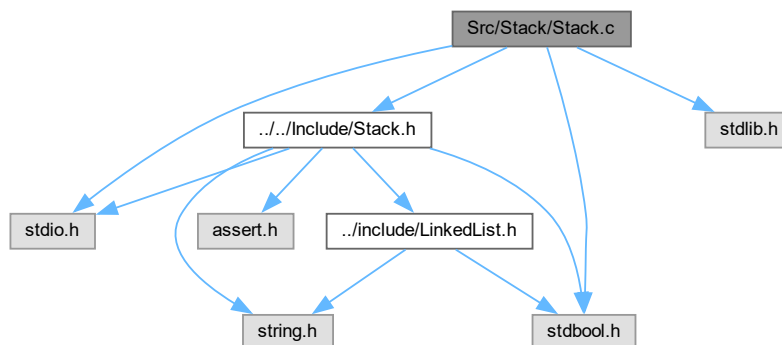
Here is the caller graph for this function:



## 5.12 Src/Stack/Stack.c File Reference

```
#include "../..//Include/Stack.h"
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
```

Include dependency graph for Stack.c:



## Functions

- `STACK * InitStack (compare Compare)`
- `bool StackIsEmpty (STACK *ourStack)`
- `void * Pop (STACK *ourStack)`
- `void Push (STACK *ourStack, void *data)`
- `void DestroyStack (STACK *ourStack)`

### 5.12.1 Detailed Description

#### Author

Tanner Ensign, Michael Vaquilar, Masaya Takahashi

#### Date

1/31/2023

### 5.12.2 Function Documentation

#### 5.12.2.1 DestroyStack()

```
void DestroyStack (  
    STACK * ourStack )
```

Destroys the stack, aka freeing the memory.

#### Parameters

<i>ourStack</i>	the stack to destroy.
-----------------	-----------------------

Here is the call graph for this function:



### 5.12.2.2 InitStack()

```
STACK * InitStack (
    compare Compare )
```

Initializes our stack and allocates memory for the queue and list.

#### Parameters

<i>Compare</i>	a compare function for the generic data type.
----------------	---

#### Returns

pointer to the stack made, NULL if it couldn't be made.

Here is the call graph for this function:



### 5.12.2.3 Pop()

```
void * Pop (
    STACK * ourStack )
```

Pop's an item off the top of the stack.

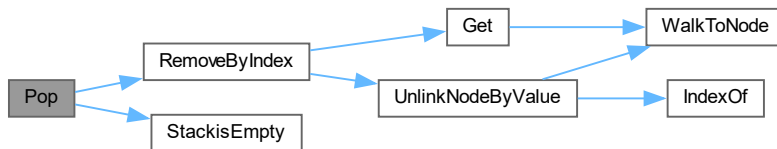
#### Parameters

<i>ourStack</i>	the stack to pop an item off of.
-----------------	----------------------------------

**Returns**

the generic pointer to the item popped off.

Here is the call graph for this function:

**5.12.2.4 Push()**

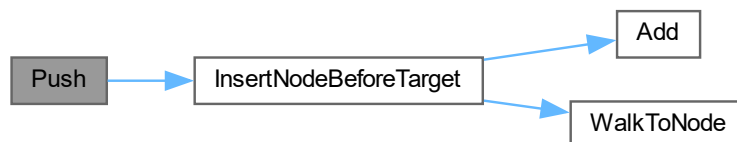
```
void Push (
    STACK * ourStack,
    void * data )
```

Pushes a generic pointer to an item onto the top of the stack.

**Parameters**

<i>ourStack</i>	the stack to push onto.
<i>data</i>	void pointer to the data to pop onto the stack.

Here is the call graph for this function:

**5.12.2.5 StackIsEmpty()**

```
bool StackIsEmpty (
    STACK * ourStack )
```

Checks if the stack is empty.

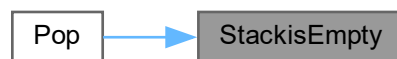
**Parameters**

<i>ourStack</i>	the stack to check if empty.
-----------------	------------------------------

**Returns**

true if empty, false otherwise.

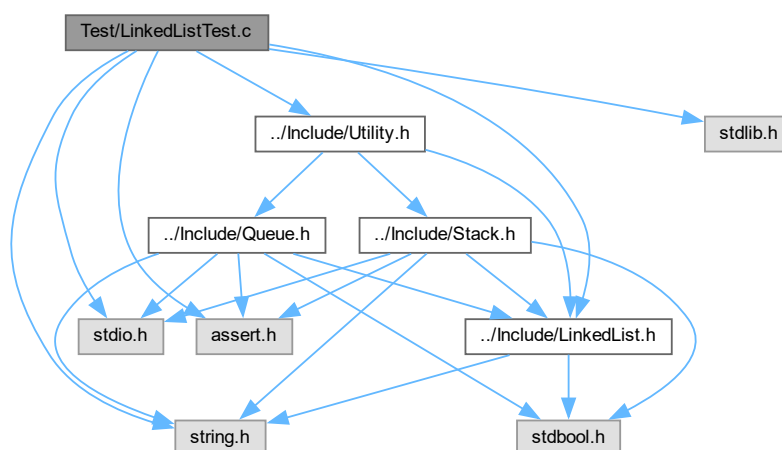
Here is the caller graph for this function:



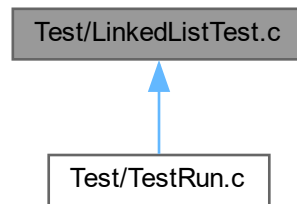
## 5.13 Test/LinkedListTest.c File Reference

```
#include <stdio.h>
#include "../Include/LinkedList.h"
#include <string.h>
#include <assert.h>
#include "../Include/Utility.h"
#include <stdlib.h>
```

Include dependency graph for `LinkedListTest.c`:



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



## Functions

- void **TestAddOne** ()
- void **TestMultipleValue** ()
- void **TestWholeList** ()
- void **TestIndexOf** ()
- void **TestIndexOfFail** ()
- void **TestWithRandomInsert** ()
- void **InsertBeforeTest** ()
- void **InsertAfterTest** ()
- void **InsertAfterTestTwo** ()
- void **removeTest** ()
- void **removeAllTest** ()
- void **TestSort** ()
- void **RunAllListTest** ()

### 5.13.1 Detailed Description

#### Author

Tanner Ensign, Michael Vaquilar, Masaya Takahashi

#### Date

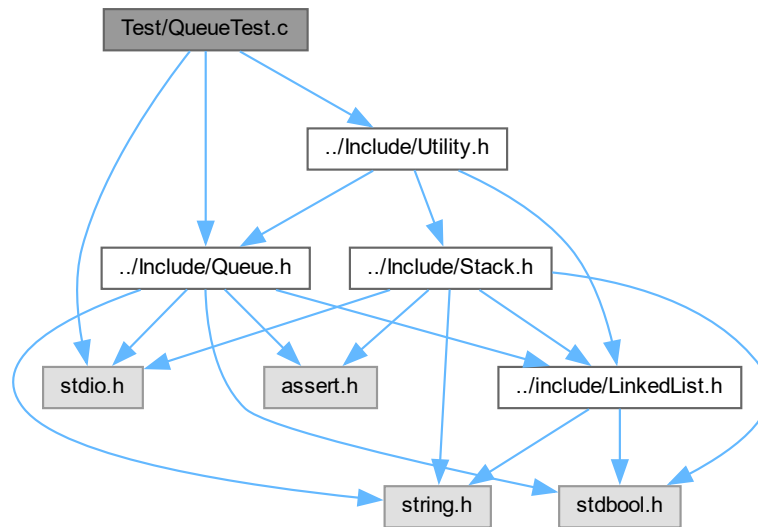
1/31/2023

## 5.14 Test/QueueTest.c File Reference

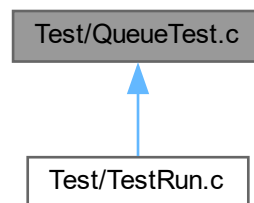
```
#include <stdio.h>
#include "../Include/Queue.h"
```

```
#include "../Include/Utility.h"
```

Include dependency graph for QueueTest.c:



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



## Functions

- void **QueueTestPushAndPop** ()
- void **QueueTestMultipleValue** ()
- void **QueueTestDequeueEmpty** ()
- void **QueueTestDequeueAll** ()
- void **RunAllQueueTest** ()

### 5.14.1 Detailed Description





## Functions

- void **StackTestPushAndPop** ()
- void **StackTestPushPopMultipleValue** ()
- void **StackTestPopEmpty** ()
- void **StackTestPopAll** ()
- void **RunAllStackTest** ()

### 5.15.1 Detailed Description

#### Author

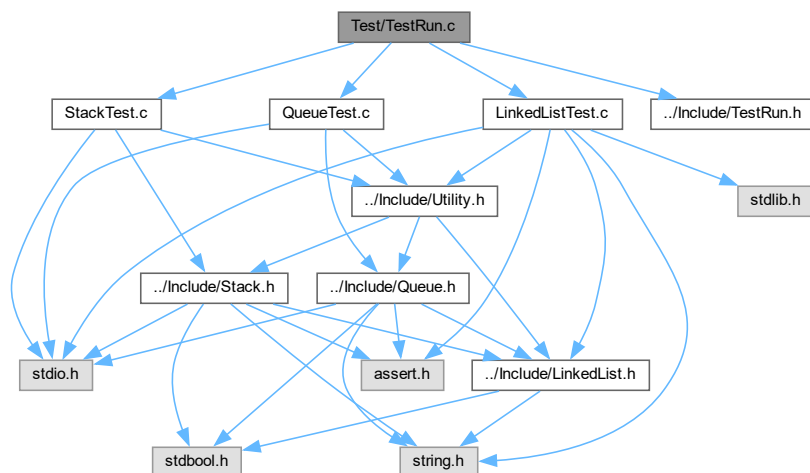
Tanner Ensign, Michael Vaquilar, Masaya Takahashi

#### Date

1/31/2023

## 5.16 Test/TestRun.c File Reference

```
#include "LinkedListTest.c"
#include "QueueTest.c"
#include "StackTest.c"
#include "../Include/TestRun.h"
Include dependency graph for TestRun.c:
```



## Functions

- void **TestAll** ()

### 5.16.1 Detailed Description

#### Author

Tanner Ensign, Michael Vaquilar, Masaya Takahashi

#### Date

1/31/2023

### 5.16.2 Function Documentation

#### 5.16.2.1 TestAll()

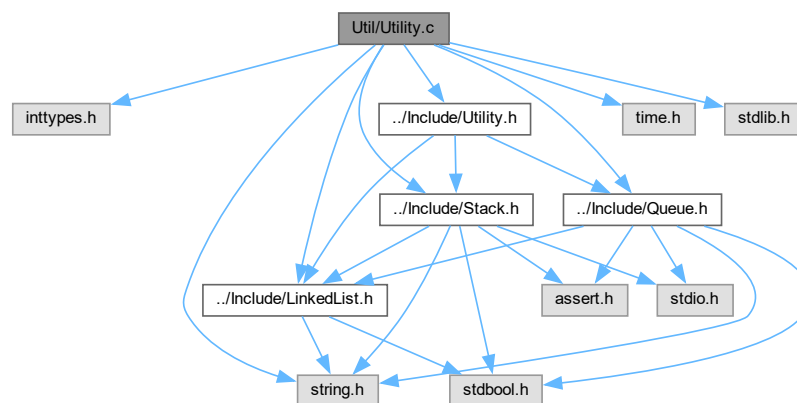
```
void TestAll ( )
```

Runs all our test.

## 5.17 Util/Utility.c File Reference

```
#include <inttypes.h>
#include <string.h>
#include "../Include/Utility.h"
#include "../Include/LinkedList.h"
#include "../Include/Queue.h"
#include "../Include/Stack.h"
#include <time.h>
#include <stdlib.h>
```

Include dependency graph for Utility.c:



## Functions

- int [compare\\_int32\\_t](#) (const void \*element1, const void \*element2)
- int [compare\\_int64\\_t](#) (const void \*element1, const void \*element2)
- int [compare\\_float](#) (const void \*element1, const void \*element2)
- int [compare\\_double](#) (const void \*element1, const void \*element2)
- int [compare\\_long\\_double](#) (const void \*element1, const void \*element2)
- int [compare\\_char](#) (const void \*element1, const void \*element2)
- int [compare\\_string](#) (const void \*element1, const void \*element2)
- int [compareIntArray](#) (int a[], int b[])
- void [TestList](#) ([LIST](#) \*listHolder, void \*expected, void \*actual, const char \*testName, bool isNULL)
- void [TestQueue](#) ([QUEUE](#) \*queue, void \*expected, void \*actual, const char \*testName, bool isNULL)
- void [TestStack](#) ([STACK](#) \*stack, void \*expected, void \*actual, const char \*testName, bool isNULL)
- int [randomInt](#) ()
- int \* [ToArray](#) ([LIST](#) \*listHolder)

### 5.17.1 Detailed Description

#### Author

Tanner Ensign, Michael Vaquilar, Masaya Takahashi

#### Date

1/31/2023

### 5.17.2 Function Documentation

#### 5.17.2.1 [compare\\_char\(\)](#)

```
int compare_char (
    const void * element1,
    const void * element2 )
```

Compares two char's to each-other.

#### Parameters

<i>element1</i>	void pointer to the first char.
<i>element2</i>	void pointer to the second char.

#### Returns

int, 1 when the first element is greater than the second, -1 when the first element is less than the second, 0 when both elements are equal.

### 5.17.2.2 `compare_double()`

```
int compare_double (
    const void * element1,
    const void * element2 )
```

Compares two double's to each-other.

#### Parameters

<i>element1</i>	void pointer to the first double.
<i>element2</i>	void pointer to the second double.

#### Returns

int, 1 when the first element is greater than the second, -1 when the first element is less than the second, 0 when both elements are equal.

### 5.17.2.3 `compare_float()`

```
int compare_float (
    const void * element1,
    const void * element2 )
```

Compares two floats to each-other.

#### Parameters

<i>element1</i>	void pointer to the first float.
<i>element2</i>	void pointer to the second float.

#### Returns

int, 1 when the first element is greater than the second, -1 when the first element is less than the second, 0 when both elements are equal.

### 5.17.2.4 `compare_int32_t()`

```
int compare_int32_t (
    const void * element1,
    const void * element2 )
```

Compares two 32 bit integers to each-other.

**Parameters**

<i>element1</i>	void pointer to the first integer.
<i>element2</i>	void pointer to the second integer.

**Returns**

int, 1 when the first element is greater than the second, -1 when the first element is less than the second, 0 when both elements are equal.

**5.17.2.5 compare\_int64\_t()**

```
int compare_int64_t (
    const void * element1,
    const void * element2 )
```

Compares two 64 bit integers to each-other.

**Parameters**

<i>element1</i>	void pointer to the first integer.
<i>element2</i>	void pointer to the second integer.

**Returns**

int, 1 when the first element is greater than the second, -1 when the first element is less than the second, 0 when both elements are equal.

**5.17.2.6 compare\_long\_double()**

```
int compare_long_double (
    const void * element1,
    const void * element2 )
```

Compares two long doubles to each-other.

**Parameters**

<i>element1</i>	void pointer to the first long double.
<i>element2</i>	void pointer to the second long double.

**Returns**

int, 1 when the first element is greater than the second, -1 when the first element is less than the second, 0 when both elements are equal.

**5.17.2.7 compare\_string()**

```
int compare_string (
    const void * element1,
    const void * element2 )
```

Compares two strings to each-other.

**Parameters**

<i>element1</i>	void pointer to the first string.
<i>element2</i>	void pointer to the second string.

**Returns**

int, 1 when the first element is greater than the second, -1 when the first element is less than the second, 0 when both elements are equal.

**5.17.2.8 compareIntArray()**

```
int compareIntArray (
    int a[],
    int b[] )
```

Compares to int arrays to see if they are similar.

**Parameters**

<i>a</i>	first int array.
<i>b</i>	second int array.

**Returns**

0 if not, 1 if equal.

**5.17.2.9 randomInt()**

```
int randomInt ( )
```

Generates a random 32 bit int.

**Returns**

randomly generated int.

**5.17.2.10 TestList()**

```
void TestList (
    LIST * listHolder,
    void * expected,
    void * actual,
    const char * testName,
    bool isNULL )
```

Small method for a list unit test, checks if two objects are equal, if so they passed.

**Parameters**

<i>listHolder</i>	linkedList to test
<i>expected</i>	output
<i>actual</i>	output
<i>testName</i>	name of the test
<i>isNULL</i>	check if actual and expected both should be NULL

**5.17.2.11 TestQueue()**

```
void TestQueue (
    QUEUE * queue,
    void * expected,
    void * actual,
    const char * testName,
    bool isNULL )
```

Small method for a list [Queue](#) test, checks if two objects are equal, if so they passed.

**Parameters**

<i>queue</i>	to test
<i>expected</i>	output
<i>actual</i>	output
<i>testName</i>	name of the test
<i>isNULL</i>	check if actual and expected both should be NULL



### 5.17.2.12 TestStack()

```
void TestStack (
    STACK * stack,
    void * expected,
    void * actual,
    const char * testName,
    bool isNULL )
```

Small method for a [Stack](#) unit test, checks if two objects are equal, if so they passed.

#### Parameters

<i>stack</i>	to test
<i>expected</i>	output
<i>actual</i>	output
<i>testName</i>	name of the test
<i>isNULL</i>	check if actual and expected both should be NULL

### 5.17.2.13 ToArray()

```
int * ToArray (
    LIST * listHolder )
```

Converts a int LinkedList into an int array

#### Parameters

<i>listHolder</i>	the list to convert into an array
-------------------	-----------------------------------

#### Returns

int array



# Index

## Add

LinkedList.c, [44](#)  
LinkedList.h, [13](#)

## compare

LinkedList.h, [12](#)

## compare\_char

Utility.c, [70](#)  
Utility.h, [37](#)

## compare\_double

Utility.c, [70](#)  
Utility.h, [38](#)

## compare\_float

Utility.c, [71](#)  
Utility.h, [38](#)

## compare\_int32\_t

Utility.c, [71](#)  
Utility.h, [39](#)

## compare\_int64\_t

Utility.c, [72](#)  
Utility.h, [39](#)

## compare\_long\_double

Utility.c, [72](#)  
Utility.h, [39](#)

## compare\_string

Utility.c, [73](#)  
Utility.h, [40](#)

## compareIntArray

Utility.c, [73](#)  
Utility.h, [40](#)

## Dequeue

Queue.c, [56](#)  
Queue.h, [27](#)

## DestroyList

LinkedList.c, [45](#)  
LinkedList.h, [14](#)

## DestroyQueue

Queue.c, [57](#)  
Queue.h, [28](#)

## DestroyStack

Stack.c, [60](#)  
Stack.h, [32](#)

## DumpList

LinkedList.c, [45](#)  
LinkedList.h, [14](#)

## Enqueue

Queue.c, [57](#)  
Queue.h, [28](#)

## findMid

LinkedList.c, [46](#)  
LinkedList.h, [15](#)

## Get

LinkedList.c, [46](#)  
LinkedList.h, [15](#)

## Include/LinkedList.h, [11](#), [24](#)

## Include/Queue.h, [25](#), [30](#)

## Include/Stack.h, [31](#), [35](#)

## Include/TestRun.h, [36](#)

## Include/Utility.h, [36](#), [43](#)

## IndexOf

LinkedList.c, [47](#)  
LinkedList.h, [16](#)

## InitList

LinkedList.c, [48](#)  
LinkedList.h, [17](#)

## InitQueue

Queue.c, [58](#)  
Queue.h, [29](#)

## InitStack

Stack.c, [60](#)  
Stack.h, [33](#)

## InsertNodeAfterTarget

LinkedList.c, [48](#)  
LinkedList.h, [17](#)

## InsertNodeBeforeTarget

LinkedList.c, [49](#)  
LinkedList.h, [18](#)

## LinkedList.c

Add, [44](#)  
DestroyList, [45](#)  
DumpList, [45](#)  
findMid, [46](#)  
Get, [46](#)  
IndexOf, [47](#)  
InitList, [48](#)  
InsertNodeAfterTarget, [48](#)  
InsertNodeBeforeTarget, [49](#)  
MergeSort, [50](#)  
RemoveByIndex, [51](#)  
Sort, [52](#)  
SortList, [53](#)  
UnlinkNodeByValue, [54](#)  
WalkToNode, [55](#)

## LinkedList.h

Add, [13](#)

- compare, 12
- DestroyList, 14
- DumpList, 14
- findMid, 15
- Get, 15
- IndexOf, 16
- InitList, 17
- InsertNodeAfterTarget, 17
- InsertNodeBeforeTarget, 18
- LIST, 13
- MergeSort, 19
- NODE, 13
- RemoveByIndex, 20
- Sort, 21
- SortList, 22
- UnlinkNodeByValue, 23
- WalkToNode, 24
- LIST
  - LinkedList.h, 13
- LList, 7
- MergeSort
  - LinkedList.c, 50
  - LinkedList.h, 19
- NODE
  - LinkedList.h, 13
- Node, 8
- Pop
  - Stack.c, 61
  - Stack.h, 33
- Push
  - Stack.c, 62
  - Stack.h, 34
- QUEUE
  - Queue.h, 27
- Queue, 8
- Queue.c
  - Dequeue, 56
  - DestroyQueue, 57
  - Enqueue, 57
  - InitQueue, 58
  - QueueisEmpty, 59
- Queue.h
  - Dequeue, 27
  - DestroyQueue, 28
  - Enqueue, 28
  - InitQueue, 29
  - QUEUE, 27
  - QueueisEmpty, 29
- QueueisEmpty
  - Queue.c, 59
  - Queue.h, 29
- randomInt
  - Utility.c, 73
  - Utility.h, 41
- RemoveByIndex
  - LinkedList.c, 51
  - LinkedList.h, 20
- Sort
  - LinkedList.c, 52
  - LinkedList.h, 21
- SortList
  - LinkedList.c, 53
  - LinkedList.h, 22
- Src/LinkedList/LinkedList.c, 43
- Src/Queue/Queue.c, 55
- Src/Stack/Stack.c, 59
- STACK
  - Stack.h, 32
- Stack, 9
- Stack.c
  - DestroyStack, 60
  - InitStack, 60
  - Pop, 61
  - Push, 62
  - StackisEmpty, 62
- Stack.h
  - DestroyStack, 32
  - InitStack, 33
  - Pop, 33
  - Push, 34
  - STACK, 32
  - StackisEmpty, 34
  - UpdateCount, 35
- StackisEmpty
  - Stack.c, 62
  - Stack.h, 34
- Test/LinkedListTest.c, 64
- Test/QueueTest.c, 65
- Test/StackTest.c, 67
- Test/TestRun.c, 68
- TestAll
  - TestRun.c, 69
- TestList
  - Utility.c, 74
  - Utility.h, 41
- TestQueue
  - Utility.c, 74
  - Utility.h, 41
- TestRun.c
  - TestAll, 69
- TestStack
  - Utility.c, 74
  - Utility.h, 42
- ToArray
  - Utility.c, 75
  - Utility.h, 42
- UnlinkNodeByValue
  - LinkedList.c, 54
  - LinkedList.h, 23
- UpdateCount

- Stack.h, [35](#)
- Util/Utility.c, [69](#)
- Utility.c
  - compare\_char, [70](#)
  - compare\_double, [70](#)
  - compare\_float, [71](#)
  - compare\_int32\_t, [71](#)
  - compare\_int64\_t, [72](#)
  - compare\_long\_double, [72](#)
  - compare\_string, [73](#)
  - compareIntArray, [73](#)
  - randomInt, [73](#)
  - TestList, [74](#)
  - TestQueue, [74](#)
  - TestStack, [74](#)
  - ToArray, [75](#)
- Utility.h
  - compare\_char, [37](#)
  - compare\_double, [38](#)
  - compare\_float, [38](#)
  - compare\_int32\_t, [39](#)
  - compare\_int64\_t, [39](#)
  - compare\_long\_double, [39](#)
  - compare\_string, [40](#)
  - compareIntArray, [40](#)
  - randomInt, [41](#)
  - TestList, [41](#)
  - TestQueue, [41](#)
  - TestStack, [42](#)
  - ToArray, [42](#)
- WalkToNode
  - LinkedList.c, [55](#)
  - LinkedList.h, [24](#)