C Data Structures Library

1.0.0

Generated by Doxygen 1.9.6

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:

Chapter 3

File Index

3.1 File List

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

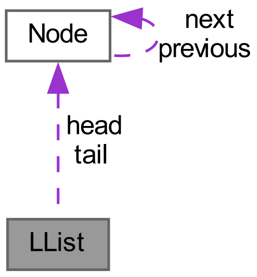
Chapter 4

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.

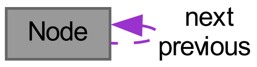
The documentation for this struct was generated from the following file:

- 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.

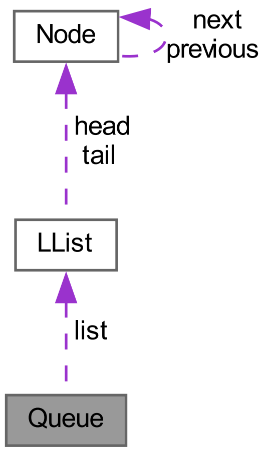
The documentation for this struct was generated from the following file:

- 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.

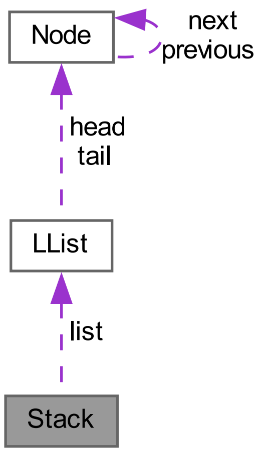
The documentation for this struct was generated from the following file:

- 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

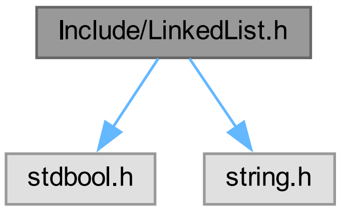
Chapter 5

File Documentation

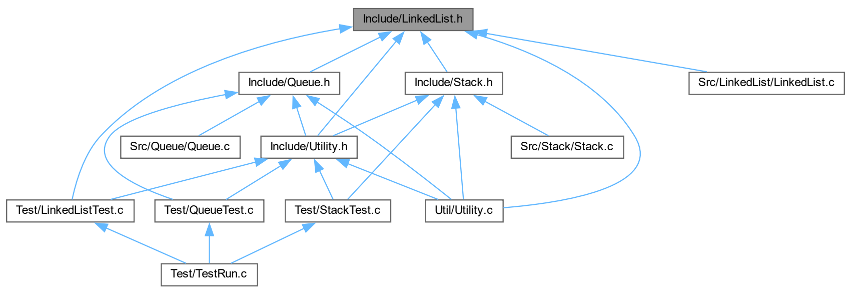
5.1 Include/LinkedList.h File Reference

```
#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()

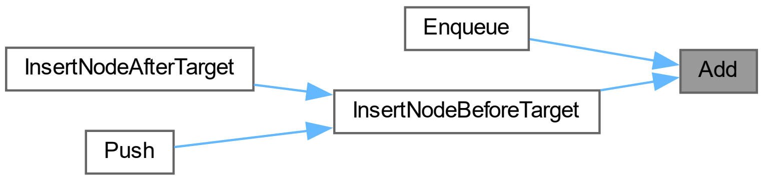
```
void Add (
            LIST * list,
            void * value )
```

Adds a value to the linked list.

Parameters

list	to Add too.
value	to be added.

Here is the caller graph for this function:



5.1.3.2 DestroyList()

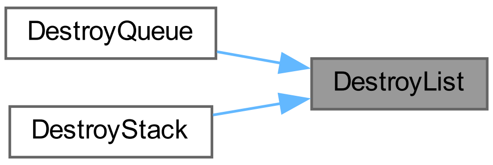
```
void DestroyList (
             LIST * list )
```

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

Parameters

list	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

list	to write to console

5.1.3.4 findMid()

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

Finds the middle of the list.

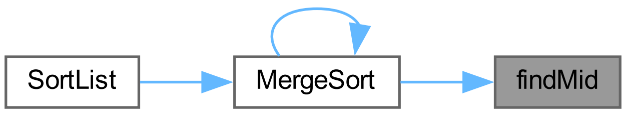
Parameters

start	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.

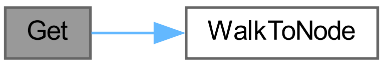
Parameters

list	to search.
index	to get value at.

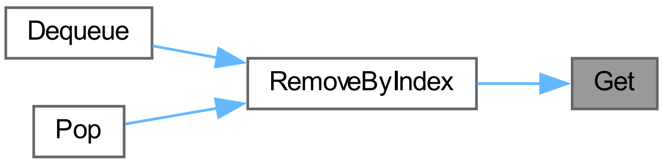
Returns

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.

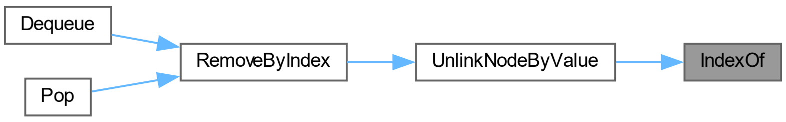
Parameters

list	to search.
value	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.

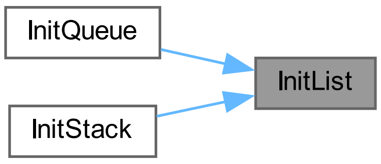
Parameters

Compare	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()

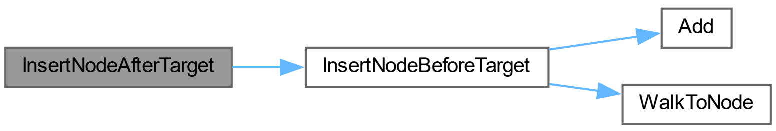
```
void InsertNodeAfterTarget (
            LIST * list,
            int index,
            void * newValue )
```

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

Parameters

list	to insert Node in.
index	to insert new value at.
newValue	the new value to insert.

Here is the call graph for this function:



5.1.3.9 InsertNodeBeforeTarget()

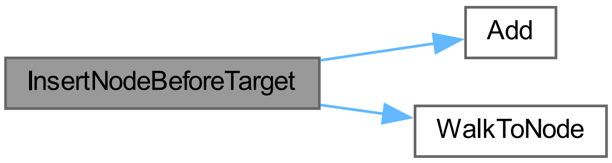
```
void InsertNodeBeforeTarget (
            LIST * list,
            int index,
            void * newValue )
```

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

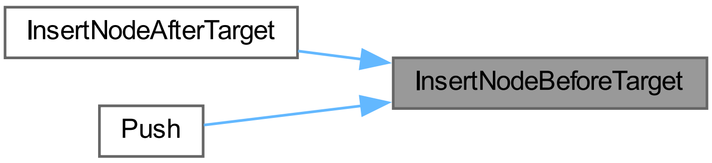
Parameters

list	to insert Node in.
index	to insert new value at.
newValue	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.

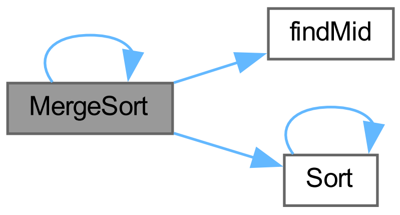
Parameters

list	to sort.
start	node

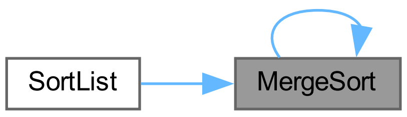
Returns

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.

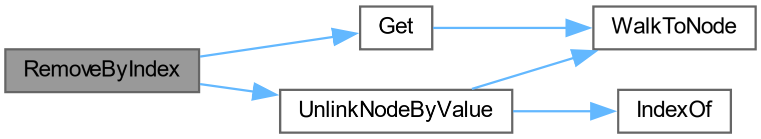
Parameters

list	to remove value from.
index	to remove

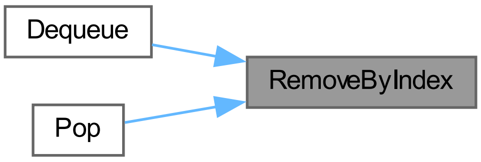
Returns

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.

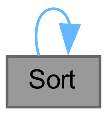
Parameters

list	that the values are from.
leftCursor	the left data value
rightCursor	the right data value

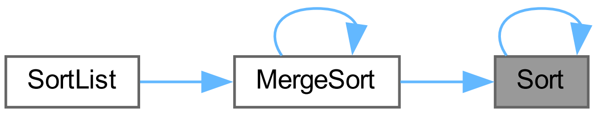
Returns

new sorted node.

Here is the call graph for this function:



Here is the caller graph for this function:



5.1.3.13 SortList()

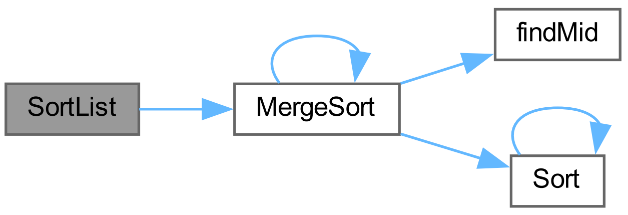
```
void SortList (
             LIST * list )
```

Calls MergeSort. Simple function for user to call

Parameters

list	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.

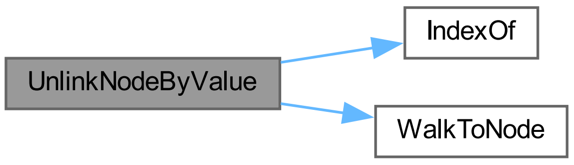
Parameters

list	to unlink Node from.
value	in the node we want to remove.

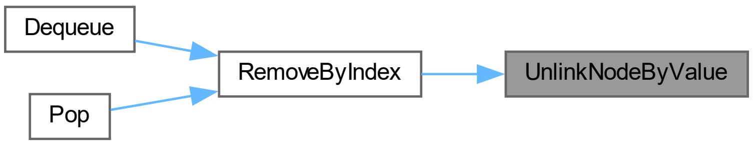
Returns

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.

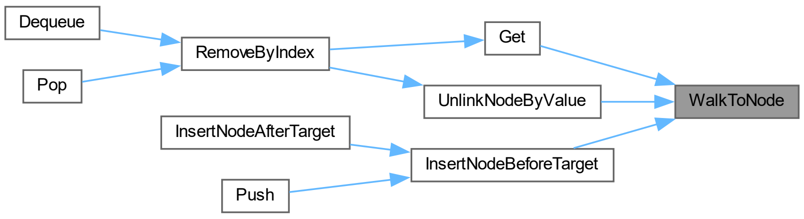
Parameters

temp	the node to walk.
location	to walk to.

Returns

NODE, the node walked to.

Here is the caller graph for this function:



5.2 LinkedList.h

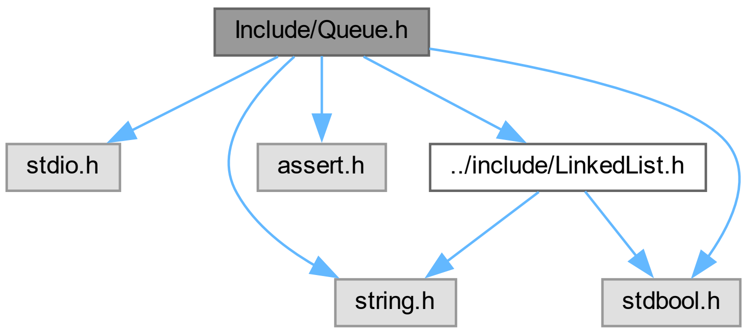
Go to the documentation of this file.

```
#ifndef DATASTRUCTURES_LINKEDLIST_H
#define DATASTRUCTURES_LINKEDLIST_H

#include <stdbool.h>
#include <string.h>

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



typedef struct Node{
    void **value;
    struct Node *next;
    struct Node *previous;
} NODE;

typedef struct LList{
    NODE *head;
    NODE *tail;
    int count;
    compare CompareTo;
} LIST;


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);

#endif //DATASTRUCTURES_LINKEDLIST_H
```

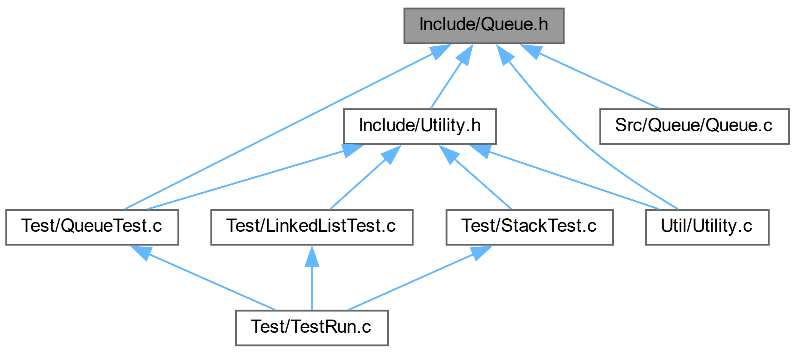
5.3 Include/Queue.h File Reference

```
#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).

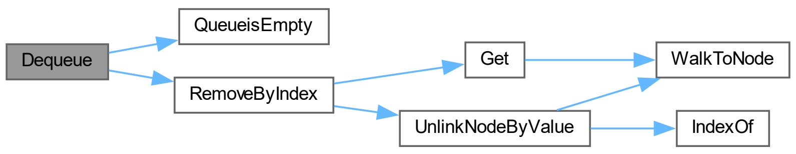
Parameters

ourQueue	the queue to dequeue an item from.

Returns

the item removed.

Here is the call graph for this function:



5.3.3.2 DestroyQueue()

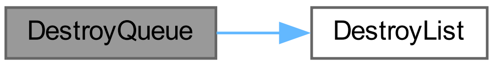
```
void DestroyQueue (
             QUEUE * ourQueue )
```

Destroys the queue, aka freeing the memory

Parameters

ourQueue	the queue to destroy.

Here is the call graph for this function:



5.3.3.3 Enqueue()

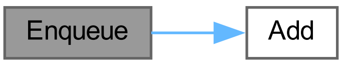
```
void Enqueue (
             QUEUE * ourQueue,
             void * item )
```

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

Parameters

ourQueue	the queue to enqueue onto.
item	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.

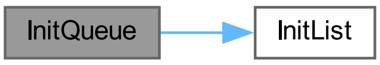
Parameters

Compare	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.

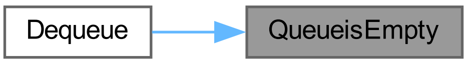
Parameters

ourQueue	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

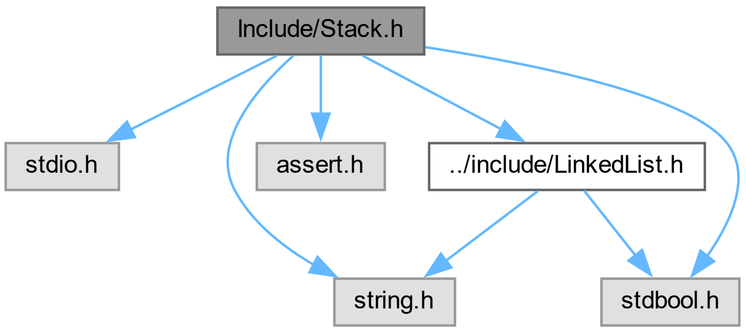
Go to the documentation of this file.

```
#ifndef DATASTRUCTURES_QUEUE_H
#define DATASTRUCTURES_QUEUE_H

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

typedef struct Queue{
    int Count;
    LIST *list;
}QUEUE;

QUEUE* InitQueue(compare Compare);

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

void* Dequeue(QUEUE *ourQueue);

bool QueueisEmpty(QUEUE *ourQueue);

void DestroyQueue(QUEUE *ourQueue);

#endif //DATASTRUCTURES_QUEUE_H
```

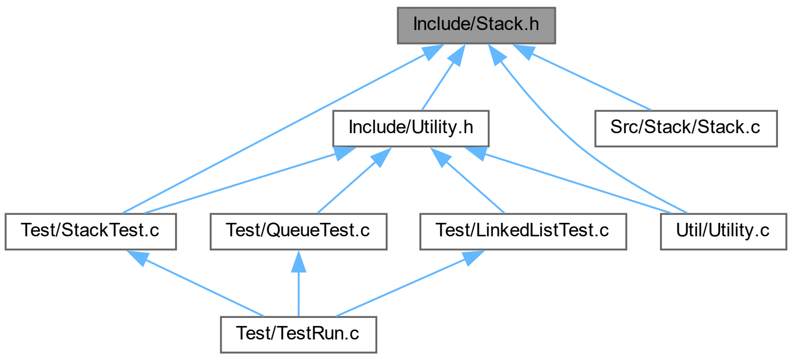
5.5 Include/Stack.h File Reference

```
#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()

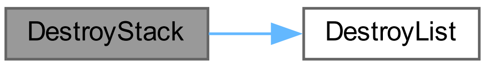
```
void DestroyStack (
            STACK ∗ ourStack )
```

Destroys the stack, aka freeing the memory.

Parameters

ourStack	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.

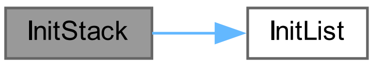
Parameters

Compare	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.

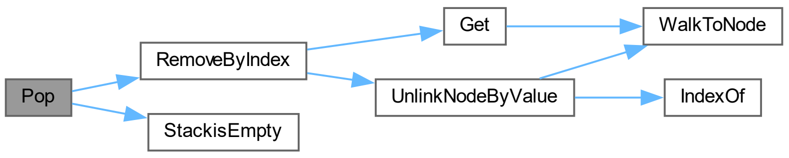
Parameters

ourStack	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()

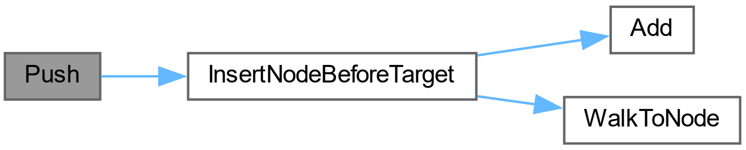
```
void Push (
             STACK * ourStack,
             void * data )
```

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

Parameters

ourStack	the stack to push onto.
data	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.

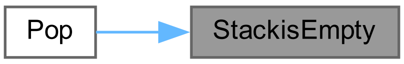
Parameters

ourStack	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.

```

#ifndef DATASTRUCTURES_STACK_H
#define DATASTRUCTURES_STACK_H

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

typedef struct Stack{
    int Count;
    LIST *list;
}STACK;

void UpdateCount();

STACK* InitStack(compare Compare);
```

```

bool StackisEmpty(STACK *ourStack);

void *Pop(STACK *ourStack);

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

void DestroyStack(STACK *ourStack);

#endif //DATASTRUCTURES_STACK_H
```

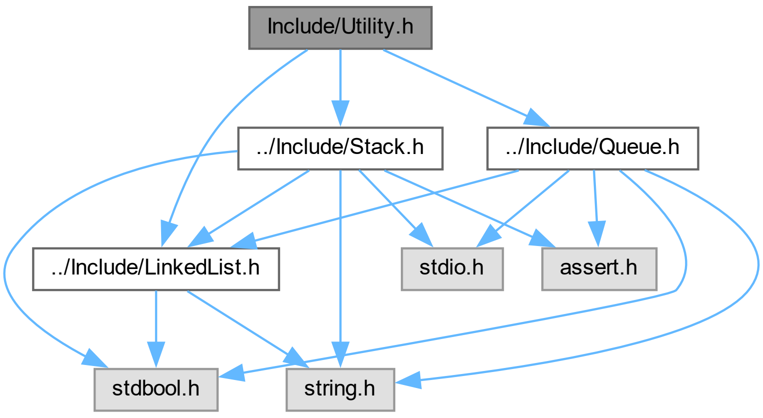
5.7 TestRun.h

```

#ifndef DATASTRUCTURES_TESTRUN_H
#define DATASTRUCTURES_TESTRUN_H

void TestAll();

#endif //DATASTRUCTURES_TESTRUN_H
```

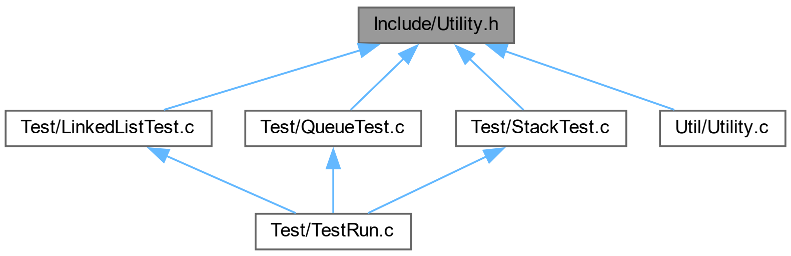
5.8 Include/Utility.h File Reference

```
#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 compareIntArrays (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

element1	void pointer to the first char.
element2	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

element1	void pointer to the first double.
element2	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

element1	void pointer to the first float.
element2	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

element1	void pointer to the first integer.
element2	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

element1	void pointer to the first integer.
element2	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

element1	void pointer to the first long double.
element2	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

element1	void pointer to the first string.
element2	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 compareIntArrays()

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

Compares to int arrays to see if they are similar.

Parameters

a	first int array.
b	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

listHolder	linkedlist to test
expected	output
actual	output
testName	name of the test
isNULL	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

queue	to test
expected	output
actual	output
testName	name of the test
isNULL	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

stack	to test
expected	output
actual	output
testName	name of the test
isNULL	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

listHolder	the list to convert into an array

Returns

int array

5.9 Utility.h

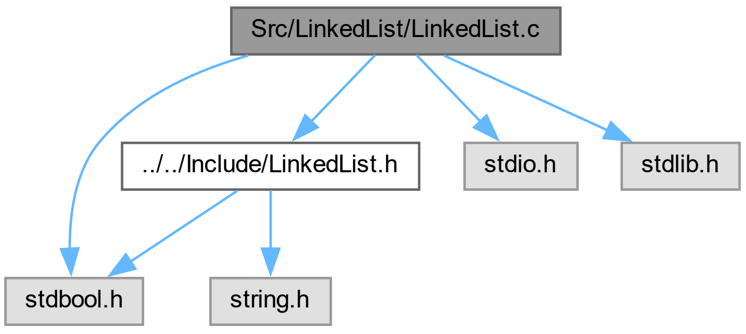
Go to the documentation of this file.

```
#ifndef DATASTRUCTURES_UTILITY_H
#define DATASTRUCTURES_UTILITY_H

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

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 compareIntArrays(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);

#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()

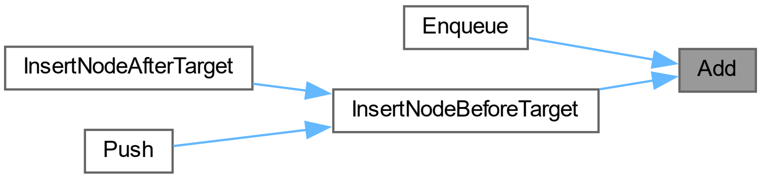
```
void Add (
            LIST * list,
            void * value )
```

Adds a value to the linked list.

Parameters

list	to Add too.
value	to be added.

Here is the caller graph for this function:



5.10.2.2 DestroyList()

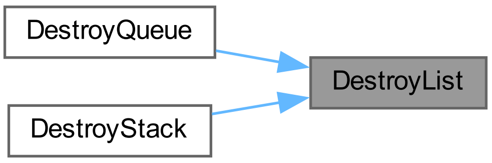
```
void DestroyList (
            LIST * list )
```

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

Parameters

list	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

list	to write to console

5.10.2.4 findMid()

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

Finds the middle of the list.

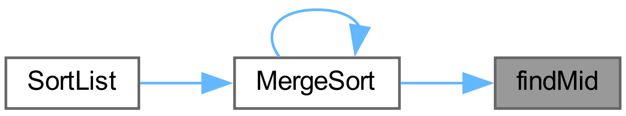
Parameters

start	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.

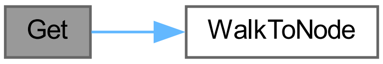
Parameters

list	to search.
index	to get value at.

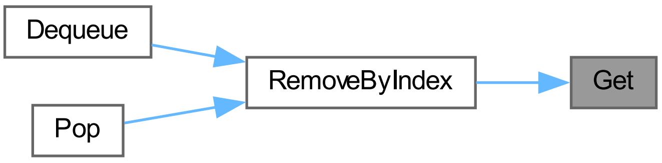
Returns

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.

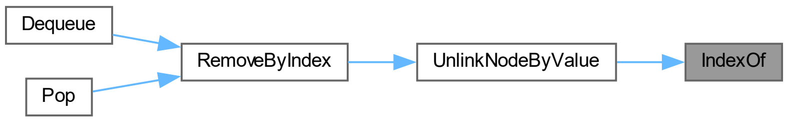
Parameters

list	to search.
value	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.

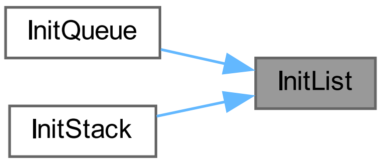
Parameters

Compare	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()

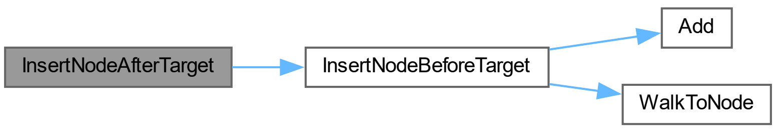
```
void InsertNodeAfterTarget (
            LIST * list,
            int index,
            void * newValue )
```

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

Parameters

list	to insert Node in.
index	to insert new value at.
newValue	the new value to insert.

Here is the call graph for this function:



5.10.2.9 InsertNodeBeforeTarget()

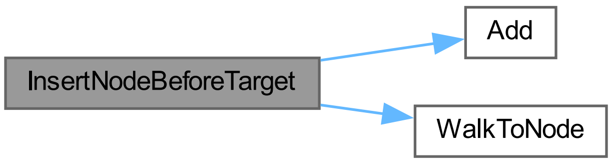
```
void InsertNodeBeforeTarget (
            LIST * list,
            int index,
            void * newValue )
```

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

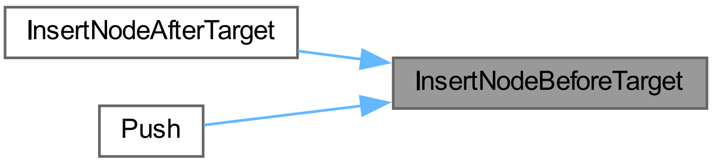
Parameters

list	to insert Node in.
index	to insert new value at.
newValue	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.

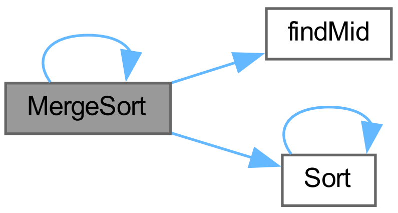
Parameters

list	to sort.
start	node

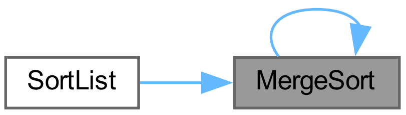
Returns

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.

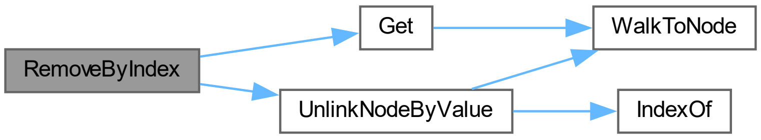
Parameters

list	to remove value from.
index	to remove

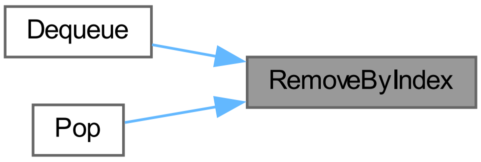
Returns

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.

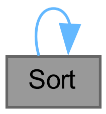
Parameters

list	that the values are from.
leftCursor	the left data value
rightCursor	the right data value

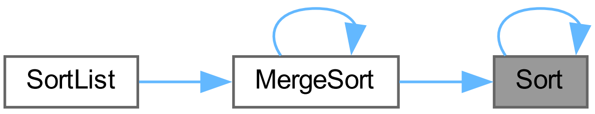
Returns

new sorted node.

Here is the call graph for this function:



Here is the caller graph for this function:



5.10.2.13 SortList()

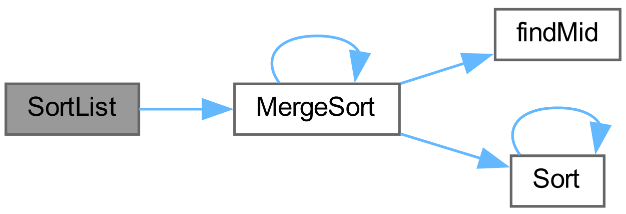
```
void SortList (
             LIST * list )
```

Calls MergeSort. Simple function for user to call

Parameters

list	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.

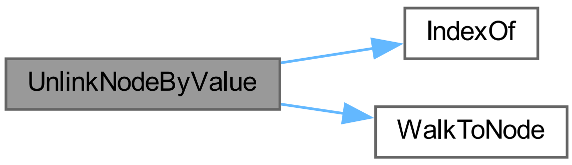
Parameters

list	to unlink Node from.
value	in the node we want to remove.

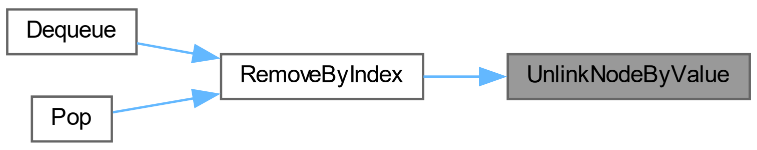
Returns

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.

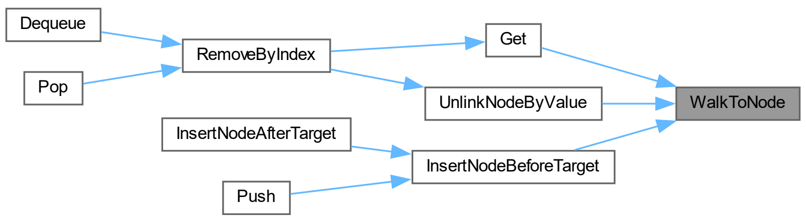
Parameters

temp	the node to walk.
location	to walk to.

Returns

NODE, the node walked to.

Here is the caller graph for this function:

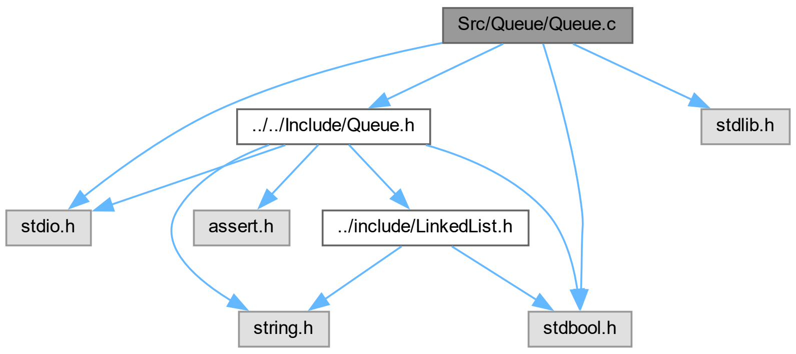


5.11 Src/Queue/Queue.c File Reference

```
#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).

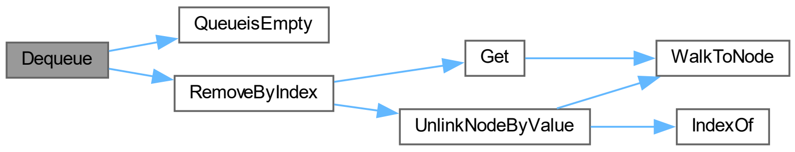
Parameters

ourQueue	the queue to dequeue an item from.

Returns

the item removed.

Here is the call graph for this function:



5.11.2.2 DestroyQueue()

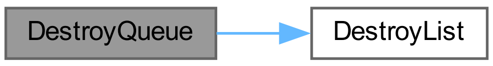
```
void DestroyQueue (
            QUEUE * ourQueue )
```

Destroys the queue, aka freeing the memory

Parameters

ourQueue	the queue to destroy.

Here is the call graph for this function:



5.11.2.3 Enqueue()

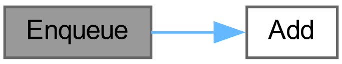
```
void Enqueue (
            QUEUE * ourQueue,
            void * item )
```

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

Parameters

ourQueue	the queue to enqueue onto.
item	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.

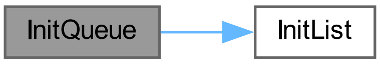
Parameters

Compare	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.

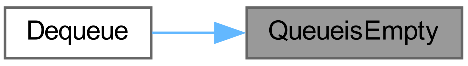
Parameters

ourQueue	the queue to check if it is empty.

Returns

true if empty, false otherwise.

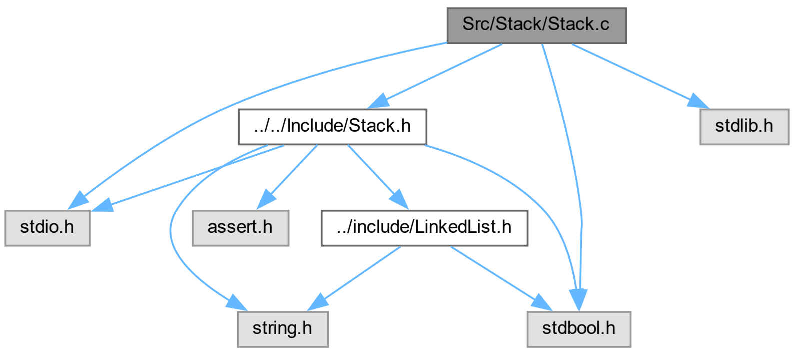
Here is the caller graph for this function:



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()

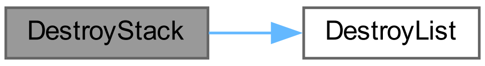
```
void DestroyStack (
             STACK ∗ ourStack )
```

Destroys the stack, aka freeing the memory.

Parameters

ourStack	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.

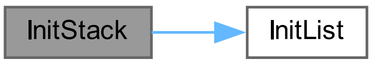
Parameters

Compare	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.

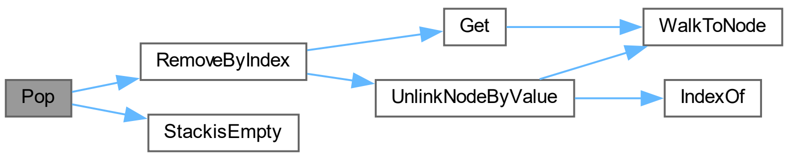
Parameters

ourStack	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()

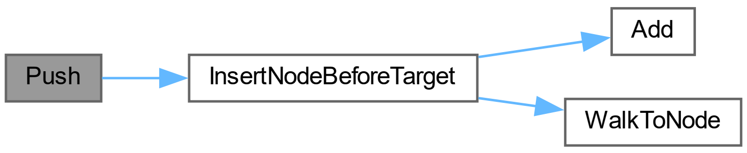
```
void Push (
            STACK * ourStack,
            void * data )
```

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

Parameters

ourStack	the stack to push onto.
data	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.

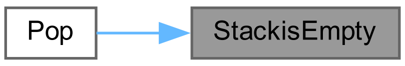
Parameters

ourStack	the stack to check if empty.

Returns

true if empty, false otherwise.

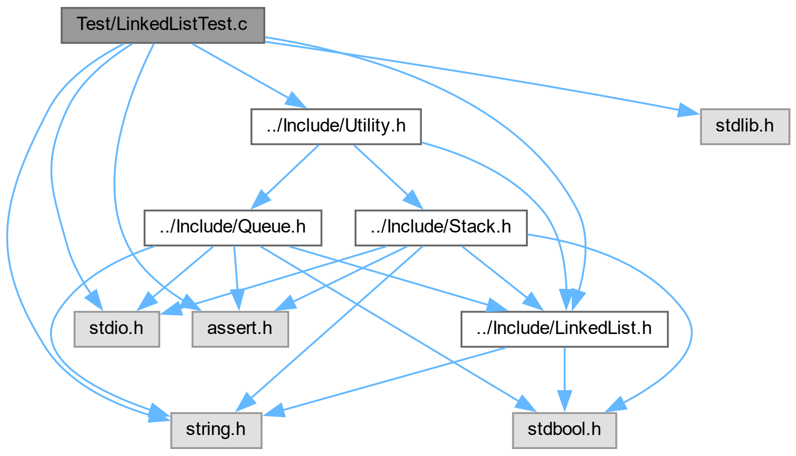
Here is the caller graph for this function:



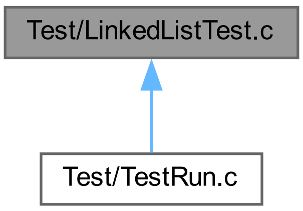
5.13 Test/LinkedListTest.c File Reference

```
#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

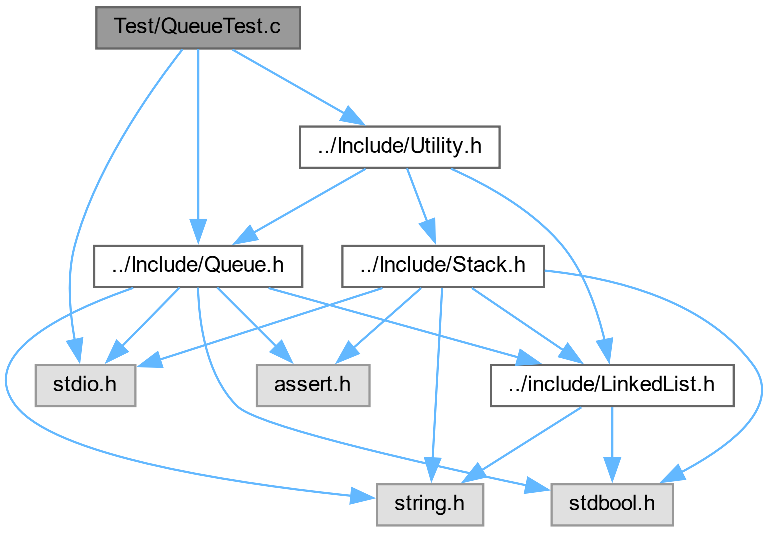
1/31/2023

5.14 Test/QueueTest.c File Reference

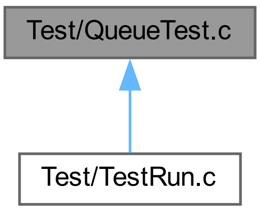
```
#include <stdio.h>
#include "../Include/Queue.h"
```

`#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

Author

Tanner Ensign, Michael Vaquilar, Masaya Takahashi

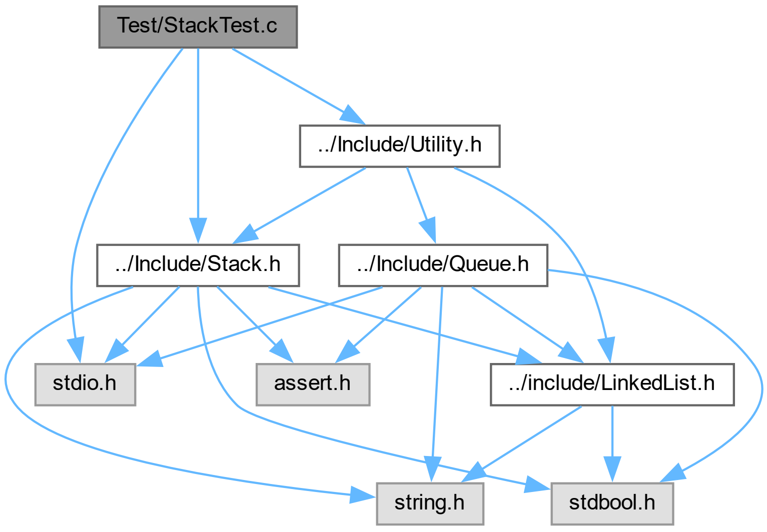
Date

1/31/2023

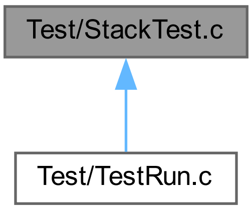
5.15 Test/StackTest.c File Reference

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

Include dependency graph for StackTest.c:



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



Functions

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

5.15.1 Detailed Description

Author

Tanner Ensign, Michael Vaquilar, Masaya Takahashi

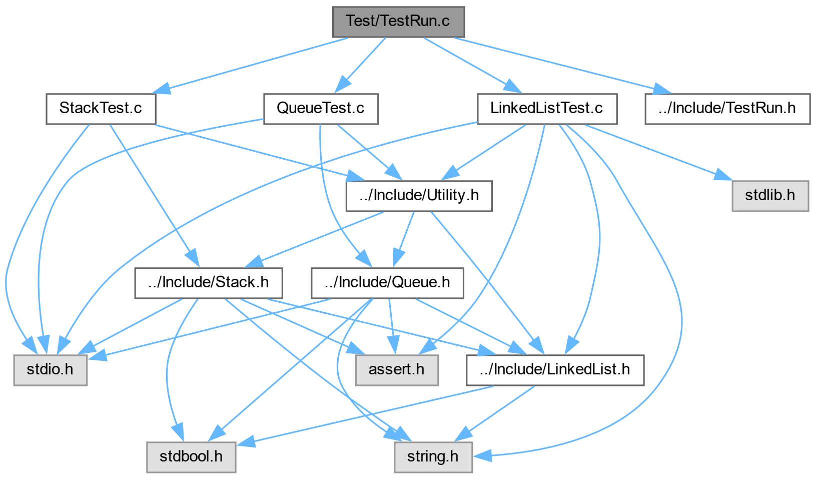
Date

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()

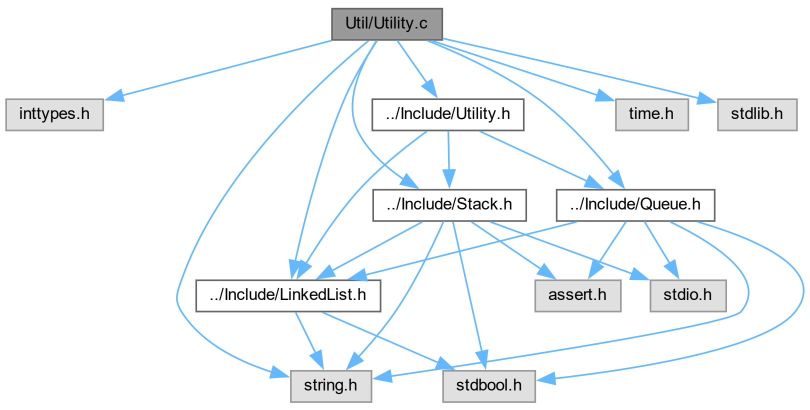
```
void TestAll ( )
```

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 compareIntArrays (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

element1	void pointer to the first char.
element2	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

element1	void pointer to the first double.
element2	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

element1	void pointer to the first float.
element2	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

element1	void pointer to the first integer.
element2	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

element1	void pointer to the first integer.
element2	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

element1	void pointer to the first long double.
element2	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

element1	void pointer to the first string.
element2	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 compareIntArrays()

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

Compares to int arrays to see if they are similar.

Parameters

a	first int array.
b	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

listHolder	linkedlist to test
expected	output
actual	output
testName	name of the test
isNULL	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

queue	to test
expected	output
actual	output
testName	name of the test
isNULL	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

stack	to test
expected	output
actual	output
testName	name of the test
isNULL	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

listHolder	the list to convert into an array

Returns

int array

Index