Data Structures Library

Generated by Doxygen 1.9.6

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 LList Struct Reference

`#include <LinkedList.h>`

Collaboration diagram for LList:

Data Fields

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

3.1.1 Detailed Description

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

3.1.2 Field Documentation

3.1.2.1 CompareTo

`compare CompareTo`

3.1.2.2 count

`int count`

3.1.2.3 head

```
NODE* head
```

3.1.2.4 tail

```
NODE* tail
```

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

- LinkedList.h

3.2 Node Struct Reference

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

Collaboration diagram for Node:

Data Fields

- void ** value
- struct Node * next
- struct Node * previous

3.2.1 Detailed Description

Node structure to hold the value along with the next and previous nodes.

3.2.2 Field Documentation

3.2.2.1 next

```
struct Node* next
```

3.2.2.2 previous

```
struct Node* previous
```

3.2.2.3 value

```
void** value
```

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

- LinkedList.h

3.3 Queue Struct Reference

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

Collaboration diagram for Queue:

Data Fields

- int Count
- LIST * list

3.3.1 Detailed Description

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

3.3.2 Field Documentation

3.3.2.1 Count

```
int Count
```

3.3.2.2 list

```
LIST* list
```

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

- Queue.h

3.4 Stack Struct Reference

`#include <Stack.h>`

Collaboration diagram for Stack:

Data Fields

- int Count
- LIST * list

3.4.1 Detailed Description

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

3.4.2 Field Documentation

3.4.2.1 Count

`int Count`

3.4.2.2 list

`LIST* list`

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

- Stack.h

Chapter 4

File Documentation

4.1 LinkedList.h File Reference

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

Include dependency graph for LinkedList.h:

4.2 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
00045 extern LIST *list;
00046
00053 LIST* InitList(compare Compare);
00054
00059 void Add(void *value);
00060
00066 void *Get(int index);
00067
00071 void DestroyList();
00072
00076 void DumpList();
00077
00083 int IndexOf(void *value);
00084
00091 bool InsertNodeBeforeTarget(int index, void *newValue);
00092
00099 bool InsertNodeAfterTarget(int index, void *newValue);
00100
```

```
00106 void *RemoveByIndex(int index);
00107
00113 bool UnlinkNodeByValue(void *value);
00114
00121 NODE *WalkToNode(NODE *temp,int location);
00122
00128 NODE *findMid(NODE *start);
00129
00136 NODE *Sort(NODE *leftCursor, NODE *rightCursor);
00137
00143 NODE *MergeSort(NODE *start);
00144
00148 void SortList();
00149
00150 #endif //DATASTRUCTURES_LINKEDLIST_H
```

4.3 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 (void *item)
- void * Dequeue ()
- bool isEmpty ()
- void DestroyQueue ()

Variables

- QUEUE * ourQueue

4.3.1 Detailed Description

Author

Tanner Ensign, Michael Vaquilar, Masaya Takahashi

Date

1/31/2023

4.3.2 Typedef Documentation

4.3.2.1 QUEUE

```
typedef struct Queue QUEUE
```

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

4.3.3 Function Documentation

4.3.3.1 Dequeue()

```
void * Dequeue ( )
```

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

Returns

the item removed.

4.3.3.2 DestroyQueue()

```
void DestroyQueue ( )
```

Destroys the queue, aka freeing the memory

4.3.3.3 Enqueue()

```
void Enqueue (
            void * item )
```

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

Parameters

item	void * item to add.

4.3.3.4 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.

4.3.3.5 isEmpty()

```
bool isEmpty ( )
```

Checks if the queue is empty.

Returns

true if empty, false otherwise.

4.3.4 Variable Documentation

4.3.4.1 ourQueue

```
QUEUE* ourQueue  [extern]
```

our queue that can be accessed externally. Aka the queue the user creates.

4.4 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
00029 extern QUEUE *ourQueue;
00030
00036 QUEUE* InitQueue(compare Compare);
00037
00042 void Enqueue(void *item);
00043
00048 void* Dequeue();
00049
00054 bool isEmpty();
00055
00059 void DestroyQueue();
00060
00061 #endif //DATASTRUCTURES_QUEUE_H
```

4.5 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 isEmpty ()
- void * Pop ()
- bool Push (void *data)
- void DestroyStack ()

Variables

- STACK * ourStack

4.5.1 Detailed Description

Author

Tanner Ensign, Michael Vaquilar, Masaya Takahashi

Date

1/31/2023

4.5.2 Typedef Documentation

4.5.2.1 STACK

```
typedef struct Stack STACK
```

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

4.5.3 Function Documentation

4.5.3.1 DestroyStack()

```
void DestroyStack ( )
```

Destroys the stack, aka freeing the memory.

4.5.3.2 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.

4.5.3.3 isEmpty()

```
bool isEmpty ( )
```

Checks if the stack is empty.

Returns

true if empty, false otherwise.

4.5.3.4 Pop()

```
void * Pop ( )
```

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

Returns

the generic pointer to the item popped off.

4.5.3.5 Push()

```
bool Push (
            void * data )
```

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

Parameters

data	void pointer to the data to pop onto the stack.

Returns

true if the item was added onto the stack, false otherwise.

4.5.3.6 UpdateCount()

```
void UpdateCount ( )
```

Updates the count of the Stack.

4.5.4 Variable Documentation

4.5.4.1 ourStack

```
STACK* ourStack  [extern]
```

Our external stack pointer that the user creates and can access. This holds all the stacks data.

4.6 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 extern STACK *ourStack;
00030
00034 void UpdateCount();
00035
00041 STACK* InitStack(compare Compare);
00042
00047 bool isEmpty();
00048
00053 void *Pop();
00054
00060 bool Push(void *data);
00061
00065 void DestroyStack();
00066
00067 #endif //DATASTRUCTURES_STACK_H
```

4.7 Utility.h File Reference

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

Include dependency graph for Utility.h:

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)
- void TestQueue (QUEUE *queue, void *expected, void *actual, const char *testName)
- void TestStack (STACK *stack, void *expected, void *actual, const char *testName)
- int randomInt ()
- int * ToArray (LIST *listHolder)

4.7.1 Detailed Description

Author

Tanner Ensign, Michael Vaquilar, Masaya Takahashi

Date

1/31/2023

4.7.2 Function Documentation

4.7.2.1 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.

4.7.2.2 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.

4.7.2.3 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.

4.7.2.4 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.

4.7.2.5 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.

4.7.2.6 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.

4.7.2.7 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.

4.7.2.8 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.

4.7.2.9 randomInt()

```
int randomInt ( )
```

Generates a random 32 bit int.

Returns

randomly generated int.

4.7.2.10 TestList()

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

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

4.7.2.11 TestQueue()

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

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

4.7.2.12 TestStack()

```
void TestStack (
            STACK * stack,
            void * expected,
```

```
            void * actual,
            const char * testName )
```

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

4.7.2.13 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

4.8 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 compareIntArrays(int a[], int b[]);
00079
00087 void TestList(LIST *listHolder, void *expected, void *actual, const char* testName);
00088
00096 void TestQueue(QUEUE *queue, void *expected, void *actual, const char* testName);
00097
```

```
00105 void TestStack(STACK *stack, void *expected, void *actual, const char* testName);
00106
00111 int randomInt();
00112
00118 int * ToArray(LIST *listHolder);
00119
00120 #endif //DATASTRUCTURES_UTILITY_H
```

Index