

## C Linked List

Generated by Doxygen 1.8.3.1

Wed Sep 25 2013 15:38:23



# Contents

<b>1</b>	<b>File Index</b>	<b>1</b>
1.1	File List . . . . .	1
<b>2</b>	<b>File Documentation</b>	<b>3</b>
2.1	LinkedList.h File Reference . . . . .	3
2.1.1	Detailed Description . . . . .	4
2.1.2	Typedef Documentation . . . . .	4
2.1.2.1	sLinkedList . . . . .	4
2.1.2.2	sListIterator . . . . .	4
2.1.3	Function Documentation . . . . .	4
2.1.3.1	listClear . . . . .	4
2.1.3.2	listEmpty . . . . .	4
2.1.3.3	listErase . . . . .	4
2.1.3.4	listGet . . . . .	5
2.1.3.5	listHead . . . . .	5
2.1.3.6	listInitialize . . . . .	5
2.1.3.7	listInsert . . . . .	6
2.1.3.8	listIteratorEnd . . . . .	6
2.1.3.9	listIteratorNext . . . . .	6
2.1.3.10	listPopBack . . . . .	6
2.1.3.11	listPopFront . . . . .	7
2.1.3.12	listPushBack . . . . .	7
2.1.3.13	listPushFront . . . . .	7
2.1.3.14	listSize . . . . .	7
	<b>Index</b>	<b>8</b>



# Chapter 1

## File Index

### 1.1 File List

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

<a href="#">LinkedList.h</a>	
A Generic Linked List implementation in C . . . . .	3



## Chapter 2

# File Documentation

### 2.1 LinkedList.h File Reference

A Generic Linked List implementation in C.

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

#### Typedefs

- typedef struct [sLinkedList](#) [sLinkedList](#)
- typedef struct [sListIterator](#) [sListIterator](#)

#### Functions

- void [listInitialize](#) ([sLinkedList](#) \*\*List, size\_t ElementSize, void(\*EraseFun)(void \*))  
*Initialize a linked list.*
- void [listPushBack](#) ([sLinkedList](#) \*List, void \*Data)  
*Insert a copy of the given data to the end of the list.*
- void [listPushFront](#) ([sLinkedList](#) \*List, void \*Data)  
*Insert a copy of the given data to the front of the list.*
- void [listInsert](#) ([sListIterator](#) \*Iterator, void \*Data)  
*Insert a copy of the given data into the list at the given iterator position.*
- void [listPopFront](#) ([sLinkedList](#) \*List)  
*Pop the first element of the list.*
- void [listPopBack](#) ([sLinkedList](#) \*List)  
*Pop the last element of the list.*
- void [listErase](#) ([sListIterator](#) \*Iterator)  
*Erase the element pointed to by Iterator.*
- void \* [listGet](#) ([sListIterator](#) \*Iterator)  
*Return the data held by an iterator.*
- void [listHead](#) ([sLinkedList](#) \*List, [sListIterator](#) \*\*It)  
*Initialize an iterator to the head of the list.*
- size\_t [listSize](#) ([sLinkedList](#) \*List)  
*Return the number of elements in a list.*
- int [listEmpty](#) ([sLinkedList](#) \*List)  
*Check whether the list is empty or contains elements.*

- void `listClear` (`sLinkedList` \*List)  
*Clear the list.*
- void `listIteratorNext` (`sListIterator` \*Iterator)  
*Advance an iterator to the next element in a list.*
- int `listIteratorEnd` (`sListIterator` \*Iterator)

### 2.1.1 Detailed Description

A Generic Linked List implementation in C. Jimmy Holm, Marcus Munger

Date

September 25, 2013

### 2.1.2 Typedef Documentation

#### 2.1.2.1 typedef struct sLinkedList sLinkedList

Linked List structure

#### 2.1.2.2 typedef struct sListIterator sListIterator

Generic Iterator for linked lists

### 2.1.3 Function Documentation

#### 2.1.3.1 void listClear ( sLinkedList \* List )

Clear the list.

Parameters

<i>List</i>	a pointer to an initialized list listClear calls listErase on every element in the list, calling upon the erasure function on each element if available.
-------------	--

#### 2.1.3.2 int listEmpty ( sLinkedList \* List )

Check whether the list is empty or contains elements.

Parameters

<i>List</i>	a pointer to an initialized list
-------------	----------------------------------

Returns

1 if the list contains no elements or 0 otherwise listEmpty returns a boolean integer based on whether the list is empty or contains elements.

#### 2.1.3.3 void listErase ( sListIterator \* Iterator )

Erase the element pointed to by Iterator.



## Parameters

<i>iterator</i>	an initialized iterator into a linked list. listErase erases the element pointed to by the iterator, removing it from its list and calling upon the data's erasure function if present.
-----------------	---

## See Also

[listPopFront\(\)](#), [listPopBack\(\)](#), and [listHead\(\)](#)

## 2.1.3.4 void\* listGet ( sListIterator \* iterator )

Return the data held by an iterator.

## Parameters

<i>iterator</i>	an initialized iterator into a linked list.
-----------------	---

## Returns

the data held by iterator listGet returns the data stored in the list element pointed to by iterator.

## See Also

[listHead\(\)](#)

## 2.1.3.5 void listHead ( sLinkedList \* List, sListIterator \*\* It )

Initialize an iterator to the head of the list.

## Parameters

<i>List</i>	a pointer to an initialized list.
<i>It</i>	a reference to an uninitialized iterator pointer. listHead initialises It to point at the first element of the given list.

## Remarks

the lifetime of the iterator is not maintained by the library. The user is responsible for freeing an initialized iterator.

## 2.1.3.6 void listInitialize ( sLinkedList \*\* List, size\_t ElementSize, void(\*) (void \*) EraseFun )

Initialize a linked list.

## Parameters

<i>List</i>	a reference to an uninitialized list pointer.
<i>ElementSize</i>	the size of a list's stored data.
<i>EraseFun</i>	a pointer to a function run on any element before its erasure

## Returns

void listInitialize is in charge of creating instances of a linked list and initializing its properties. The List parameter should point to null when passed, and will point to a valid, initialized list at the return of the function. The parameter ElementSize contains the size of a given data element, and all data passed to this list is assumed

to be of this size. EraseFun allows for a special destructor function to be called on the list's elements upon erasure.

#### Remarks

The lifetime of the list is not maintained by the library; the user is responsible for freeing the List pointer when finished with it.

#### 2.1.3.7 void listInsert ( sListIterator \* Iterator, void \* Data )

Insert a copy of the given data into the list at the given iterator position.

##### Parameters

<i>Iterator</i>	pointer to an initialized iterator into a list, where the new element is to be inserted.
<i>Data</i>	a pointer to the data to be copied into the list. This functions inserts a copy of the provided data into the list in front of the current iterator position. Note that it's a <i>copy</i> of the Data parameter that is stored; the linked list does not maintain the lifetime of the original data passed.

#### 2.1.3.8 int listIteratorEnd ( sListIterator \* Iterator )

Check whether or not an iterator is at the end of its list.

##### Parameters

<i>Iterator</i>	an initialized iterator into an initialized list.
-----------------	---

##### Returns

1 if Iterator has reached the end of its list, 0 otherwise. listIteratorEnd returns a boolean integer based on whether the given iterator has reached the end of its associated list.

#### 2.1.3.9 void listIteratorNext ( sListIterator \* Iterator )

Advance an iterator to the next element in a list.

##### Parameters

<i>Iterator</i>	an initialized iterator into an initialized list. After a call to listIteratorNext, Iterator will point to the next element in its associated list.
-----------------	---

#### 2.1.3.10 void listPopBack ( sLinkedList \* List )

Pop the last element of the list.

##### Parameters

<i>List</i>	a pointer to a list previously initialized by listInitialized, from which the final element is to be removed. listPopBack removes the final element of the list, calling upon the list's erasure function if present.
-------------	---

## See Also

[listPopFront\(\)](#), [listErase\(\)](#) and [listInitialize\(\)](#)

**2.1.3.11 void listPopFront ( sLinkedList \* List )**

Pop the first element of the list.

## Parameters

<i>List</i>	a pointer to a list previously initialized by listInitialized, from which the first element is to be removed. listPopFront removes the first element of the list, calling upon the list's erasure function if present.
-------------	--

## See Also

[listPopBack\(\)](#), [listErase\(\)](#) and [listInitialize\(\)](#)

**2.1.3.12 void listPushBack ( sLinkedList \* List, void \* Data )**

Insert a copy of the given data to the end of the list.

## Parameters

<i>List</i>	a pointer to a list previously initialized with listInitialized, into which Data is to be added.
<i>Data</i>	a pointer to the data to be copied into the list.

## Returns

void listPushBack inserts a copy of the provided data into the list at the very end. Note that it's a *copy* of the Data parameter that is stored; the linked list does not maintain the lifetime of the original data passed.

## See Also

[listPushFront\(\)](#), [listInsert\(\)](#) and [listInitialize\(\)](#)

**2.1.3.13 void listPushFront ( sLinkedList \* List, void \* Data )**

Insert a copy of the given data to the front of the list.

## Parameters

<i>List</i>	a pointer to a list previously initialized with listInitialized, into which Data is to be added.
<i>Data</i>	a pointer to the data to be copied into the list. listPushFront inserts a copy of the provided data into the list at the very front. Note that it's a <i>copy</i> of the Data parameter that is stored; the linked list does not maintain the lifetime of the original data passed.

## See Also

[listPushBack\(\)](#), [listInsert\(\)](#) and [listInitialize\(\)](#)

**2.1.3.14 size\_t listSize ( sLinkedList \* List )**

Return the number of elements in a list.

**Parameters**

<i>List</i>	a pointer to an initialized list
-------------	----------------------------------

**Returns**

the number of elements in the given list `listSize` returns the number of elements in the given list.

# Index

- LinkedList.h, [3](#)
  - listClear, [4](#)
  - listEmpty, [4](#)
  - listErase, [4](#)
  - listGet, [5](#)
  - listHead, [5](#)
  - listInitialize, [5](#)
  - listInsert, [6](#)
  - listIteratorEnd, [6](#)
  - listIteratorNext, [6](#)
  - listPopBack, [6](#)
  - listPopFront, [7](#)
  - listPushBack, [7](#)
  - listPushFront, [7](#)
  - listSize, [7](#)
  - sLinkedList, [4](#)
  - sListIterator, [4](#)
- listClear
  - LinkedList.h, [4](#)
- listEmpty
  - LinkedList.h, [4](#)
- listErase
  - LinkedList.h, [4](#)
- listGet
  - LinkedList.h, [5](#)
- listHead
  - LinkedList.h, [5](#)
- listInitialize
  - LinkedList.h, [5](#)
- listInsert
  - LinkedList.h, [6](#)
- listIteratorEnd
  - LinkedList.h, [6](#)
- listIteratorNext
  - LinkedList.h, [6](#)
- listPopBack
  - LinkedList.h, [6](#)
- listPopFront
  - LinkedList.h, [7](#)
- listPushBack
  - LinkedList.h, [7](#)
- listPushFront
  - LinkedList.h, [7](#)
- listSize
  - LinkedList.h, [7](#)
- sLinkedList
  - LinkedList.h, [4](#)
- sListIterator
  - LinkedList.h, [4](#)