

Singly Linked List

Generated by Doxygen 1.9.0

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 Node< T > Struct Template Reference	5
3.1.1 Detailed Description	5
3.1.2 Constructor & Destructor Documentation	6
3.1.2.1 Node() [1/2]	6
3.1.2.2 Node() [2/2]	6
3.1.2.3 ~Node()	6
3.1.3 Member Data Documentation	6
3.1.3.1 data	6
3.1.3.2 next	6
3.2 SLinkedList< T > Class Template Reference	7
3.2.1 Detailed Description	8
3.2.2 Constructor & Destructor Documentation	8
3.2.2.1 SLinkedList() [1/3]	8
3.2.2.2 SLinkedList() [2/3]	8
3.2.2.3 SLinkedList() [3/3]	9
3.2.2.4 ~SLinkedList()	9
3.2.3 Member Function Documentation	9
3.2.3.1 addFirst()	9
3.2.3.2 addLast()	10
3.2.3.3 clear()	10
3.2.3.4 empty()	10
3.2.3.5 insert()	11
3.2.3.6 operator=() [1/2]	11
3.2.3.7 operator=() [2/2]	12
3.2.3.8 operator==()	12
3.2.3.9 operator[]()	13
3.2.3.10 peek() [1/2]	13
3.2.3.11 peek() [2/2]	14
3.2.3.12 pop() [1/2]	15
3.2.3.13 pop() [2/2]	15
3.2.3.14 remove()	16
3.2.3.15 size()	16
4 File Documentation	17
4.1 SLinkedList.hpp File Reference	17
4.1.1 Detailed Description	17

4.1.2 Function Documentation	18
4.1.2.1 operator<<()	18
Index	19

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Node< T >	
The Node struct is meant to hold the data and pointer to the next list element	5
SLinkedList< T >	
A generic Singly Linked List class	7

Chapter 2

File Index

2.1 File List

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

SLinkedList.hpp	
A generic Singly Linked List data structure	17

Chapter 3

Class Documentation

3.1 Node< T > Struct Template Reference

The `Node` struct is meant to hold the data and pointer to the next list element.

```
#include <SLinkedList.hpp>
```

Public Member Functions

- `Node` (const T &`data`)
- `Node` (T &&`data`)
- `~Node` ()

Public Attributes

- T `data`
- `Node< T > * next`

Friends

- `std::ostream & operator<< (std::ostream &output, const Node< T > &node)`

3.1.1 Detailed Description

```
template<typename T>
struct Node< T >
```

The `Node` struct is meant to hold the data and pointer to the next list element.

Template Parameters

<code>T</code>	Any data type or class.
----------------	-------------------------

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Node() [1/2]

```
template<typename T >
Node< T >::Node (
    const T & data ) [inline]
```

Copy Constructor.

3.1.2.2 Node() [2/2]

```
template<typename T >
Node< T >::Node (
    T && data ) [inline]
```

Move Constructor.

3.1.2.3 ~Node()

```
template<typename T >
Node< T >::~~Node ( ) [inline]
```

Struct Destructor.

3.1.3 Member Data Documentation

3.1.3.1 data

```
template<typename T >
T Node< T >::data
```

The data.

3.1.3.2 next

```
template<typename T >
Node<T>* Node< T >::next
```

Pointer to the next node in the list.

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

- [SLinkedList.hpp](#)

3.2 SLinkedList< T > Class Template Reference

A generic Singly Linked List class.

```
#include <SLinkedList.hpp>
```

Public Member Functions

- [SLinkedList](#) ()
Default Constructor.
- [SLinkedList](#) (const [SLinkedList](#)< T > ©List)
Copy Constructor.
- [SLinkedList](#) ([SLinkedList](#)< T > &&moveList)
Move Constructor.
- [~SLinkedList](#) ()
Class Destructor.
- void [addFirst](#) (const T data)
Adds data to the front of the list.
- void [addLast](#) (const T data)
Adds data to the end of the list.
- void [insert](#) (const T data, const int index)
Inserts data into the list at the specified index.
- void [remove](#) (const int index)
Removes data from the list at the specified index.
- void [clear](#) ()
Clears the entire list recursively and resets all field elements to default.
- T [pop](#) ()
Removes and returns the head of the list.
- T [pop](#) (const int index)
Removes and returns data from the list at the specified index.
- T [peek](#) ()
Returns, but does not remove, the head of the list.
- T [peek](#) (const int index)
Returns, but does not remove, data from the list at the specified index.
- int [size](#) ()
Returns the size of the list.
- bool [empty](#) ()
Returns true if the list is empty and false otherwise.
- T & [operator\[\]](#) (const int index)
Subscript operator.
- bool [operator==](#) (const [SLinkedList](#) &compareList)
Equality comparison operator.
- [SLinkedList](#)< T > & [operator=](#) (const [SLinkedList](#) ©List)
Copy assignment operator.
- [SLinkedList](#)< T > & [operator=](#) ([SLinkedList](#) &&moveList)
Move assignment operator.

Friends

- template<typename Type >
std::ostream & [operator](#)<< (std::ostream &output, const [SLinkedList](#)< Type > &list)

3.2.1 Detailed Description

```
template<typename T>
class SLinkedList< T >
```

A generic Singly Linked List class.

This Singly Linked List is templated to use any data type or class.

Template Parameters

<i>T</i>	Any data type or class.
----------	-------------------------

3.2.2 Constructor & Destructor Documentation

3.2.2.1 SLinkedList() [1/3]

```
template<typename T >
SLinkedList< T >::SLinkedList
```

Default Constructor.

Template Parameters

<i>T</i>	Any data type or class.
----------	-------------------------

Initializes this linked list object with a nullptr head and tail and size of 0;

3.2.2.2 SLinkedList() [2/3]

```
template<typename T >
SLinkedList< T >::SLinkedList (
    const SLinkedList< T > & copyList )
```

Copy Constructor.

Template Parameters

<i>T</i>	Any data type or class.
----------	-------------------------

Parameters

<i>copyList</i>	The list whose contents will be copied into this linked list object.
-----------------	--

3.2.2.3 SLinkedList() [3/3]

```
template<typename T >
SLinkedList< T >::SLinkedList (
    SLinkedList< T > && moveList )
```

Move Constructor.

Template Parameters

<i>T</i>	Any data type or class.
----------	-------------------------

Parameters

<i>moveList</i>	The list whose contents will be moved into this linked list object.
-----------------	---

Note

std::move() needs to be used to call this constructor.

3.2.2.4 ~SLinkedList()

```
template<typename T >
SLinkedList< T >::~~SLinkedList
```

Class Destructor.

Template Parameters

<i>T</i>	Any data type or class.
----------	-------------------------

Recursively clears the list using [clear\(\)](#) function.

3.2.3 Member Function Documentation

3.2.3.1 addFirst()

```
template<typename T >
void SLinkedList< T >::addFirst (
    const T data )
```

Adds data to the front of the list.

Template Parameters

<i>T</i>	Any data type or class.
----------	-------------------------

Parameters

<i>data</i>	The data you want to add to the list.
-------------	---------------------------------------

3.2.3.2 addLast()

```
template<typename T >
void SLinkedList< T >::addLast (
    const T data )
```

Adds data to the end of the list.

Template Parameters

<i>T</i>	Any data type or class.
----------	-------------------------

Parameters

<i>data</i>	The data you want to add to the list.
-------------	---------------------------------------

3.2.3.3 clear()

```
template<typename T >
void SLinkedList< T >::clear
```

Clears the entire list recursively and resets all field elements to default.

Template Parameters

<i>T</i>	Any data type or class.
----------	-------------------------

3.2.3.4 empty()

```
template<typename T >
bool SLinkedList< T >::empty
```

Returns true if the list is empty and false otherwise.

Template Parameters

<i>T</i>	Any data type or class.
----------	-------------------------

Returns

A boolean flag.

3.2.3.5 insert()

```
template<typename T >
void SLinkedList< T >::insert (
    const T data,
    const int index )
```

Inserts data into the list at the specified index.

Template Parameters

<i>T</i>	Any data type or class.
----------	-------------------------

Parameters

<i>data</i>	The data you want to add to the list.
<i>index</i>	Index at which to add the data into the list.

3.2.3.6 operator=() [1/2]

```
template<typename T >
SLinkedList< T > & SLinkedList< T >::operator= (
    const SLinkedList< T > & copyList )
```

Copy assignment operator.

Template Parameters

<i>T</i>	Any data type or class.
----------	-------------------------

Parameters

<i>copyList</i>	The linked list object from which to copy elements.
-----------------	---

Returns

A reference to a copied linked list object.

Copies a linked list with the help of the copy constructor and a custom swap function.

3.2.3.7 operator=() [2/2]

```
template<typename T >
SLinkedList< T > & SLinkedList< T >::operator= (
    SLinkedList< T > && moveList )
```

Move assignment operator.

Template Parameters

<i>T</i>	Any data type or class.
----------	-------------------------

Parameters

<i>moveList</i>	The linked list object from which to move elements.
-----------------	---

Returns

A reference to a moved linked list object.

Moves linked list elements from the provided list into this linked list object. The provided linked list object is empty after the move is complete.

Note

std::move() needs to be used to call this operator.

3.2.3.8 operator==()

```
template<typename T >
bool SLinkedList< T >::operator== (
    const SLinkedList< T > & compareList )
```

Equality comparison operator.

Template Parameters

<i>T</i>	Any data type or class.
----------	-------------------------

Parameters

<i>compareList</i>	The linked list object with which to compare this linked list object.
--------------------	---

Returns

A boolean flag.

Returns true only if the objects are either the same object or both objects have the exact same elements in the exact same index locations. If the elements are the same but in different index locations, the objects are not considered similar. A false flag is returned for all other outcomes.

3.2.3.9 operator[]()

```
template<typename T >
T & SLinkedList< T >::operator[] (
    const int index )
```

Subscript operator.

Template Parameters

<i>T</i>	Any data type or class.
----------	-------------------------

Parameters

<i>index</i>	Index at which to retrieve data reference from the list.
--------------	--

Returns

The reference to the data located at the specified index in the list.

Exceptions

<i>std::out_of_range</i>	
--------------------------	--

Warning

Throws an Out Of Range exception if the list is empty or index is out of range when function is called.

Works the same way that the [] subscript operator does for arrays.

3.2.3.10 peek() [1/2]

```
template<typename T >
T SLinkedList< T >::peek
```

Returns, but does not remove, the head of the list.

Template Parameters

<i>T</i>	Any data type or class.
----------	-------------------------

Returns

The data located at the head of the list.

Exceptions

<i>std::out_of_range</i>	
--------------------------	--

Warning

Throws an Out Of Range exception if the list is empty when function is called.

3.2.3.11 peek() [2/2]

```
template<typename T >
T SLinkedList< T >::peek (
    const int index )
```

Returns, but does not remove, data from the list at the specified index.

Template Parameters

<i>T</i>	Any data type or class.
----------	-------------------------

Parameters

<i>index</i>	Index at which to retrieve data from the list.
--------------	--

Returns

The data located at the specified index of the list.

Exceptions

<i>std::out_of_range</i>	
--------------------------	--

Warning

Throws an Out Of Range exception if the list is empty or index is out of range when function is called.

3.2.3.12 pop() [1/2]

```
template<typename T >
T SLinkedList< T >::pop
```

Removes and returns the head of the list.

Template Parameters

<i>T</i>	Any data type or class.
----------	-------------------------

Returns

The removed data.

Exceptions

<i>std::out_of_range</i>	
--------------------------	--

Warning

Throws an Out Of Range exception if the list is empty when function is called.

3.2.3.13 pop() [2/2]

```
template<typename T >
T SLinkedList< T >::pop (
    const int index )
```

Removes and returns data from the list at the specified index.

Template Parameters

<i>T</i>	Any data type or class.
----------	-------------------------

Parameters

<i>index</i>	Index at which to retrieve and remove data from the list.
--------------	---

Returns

The removed data.

Exceptions

<i>std::out_of_range</i>	
--------------------------	--

Warning

Throws an Out Of Range exception if the list is empty or index is out of range when function is called.

3.2.3.14 remove()

```
template<typename T >
void SLinkedList< T >::remove (
    const int index )
```

Removes data from the list at the specified index.

Template Parameters

<i>T</i>	Any data type or class.
----------	-------------------------

Parameters

<i>index</i>	Index at which to remove data from the list.
--------------	--

3.2.3.15 size()

```
template<typename T >
int SLinkedList< T >::size
```

Returns the size of the list.

Template Parameters

<i>T</i>	Any data type or class.
----------	-------------------------

Returns

The size of the list.

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

- [SLinkedList.hpp](#)

Chapter 4

File Documentation

4.1 SLinkedList.hpp File Reference

A generic Singly Linked List data structure.

```
#include <iostream>
#include <stdexcept>
```

Classes

- struct [Node< T >](#)
The [Node](#) struct is meant to hold the data and pointer to the next list element.
- class [SLinkedList< T >](#)
A generic Singly Linked List class.

Functions

- template<typename T >
std::ostream & [operator<<](#) (std::ostream &output, const [SLinkedList< T >](#) &list)
Output stream operator.

4.1.1 Detailed Description

A generic Singly Linked List data structure.

Copyright © 2020 AI Timofeyev. All rights reserved.

Author

AI Timofeyev

Date

December 23, 2020

Version: 1.0 Modified By: Modified Date:

4.1.2 Function Documentation

4.1.2.1 operator<<()

```
template<typename T >
std::ostream& operator<< (
    std::ostream & output,
    const SLinkedList< T > & list )
```

Output stream operator.

Template Parameters

<i>T</i>	Any data type or class.
----------	-------------------------

Parameters

<i>output</i>	The output stream (usually std::cout).
<i>list</i>	The linked list object that will be printed.

Prints the list elements to the specified output stream.

Note

Any class or data type used with this linked list class MUST implement its own operator<< in order for this operator<< to work correctly. All primitive data types (int, float, double, char, string, bool) already have this operator functionality so no implementation for them is needed. But any custom class that is used with this linked list class NEEDS to implement its own operator<<.

Index

`~Node`
 `Node< T >`, 6
`~SLinkedList`
 `SLinkedList< T >`, 9

`addFirst`
 `SLinkedList< T >`, 9
`addLast`
 `SLinkedList< T >`, 10

`clear`
 `SLinkedList< T >`, 10

`data`
 `Node< T >`, 6

`empty`
 `SLinkedList< T >`, 10

`insert`
 `SLinkedList< T >`, 11

`next`
 `Node< T >`, 6
`Node`
 `Node< T >`, 6
`Node< T >`, 5
 `~Node`, 6
 `data`, 6
 `next`, 6
 `Node`, 6

`operator<<`
 `SLinkedList.hpp`, 18
`operator=`
 `SLinkedList< T >`, 11, 12
`operator==`
 `SLinkedList< T >`, 12
`operator[]`
 `SLinkedList< T >`, 13

`peek`
 `SLinkedList< T >`, 13, 14
`pop`
 `SLinkedList< T >`, 14, 15

`remove`
 `SLinkedList< T >`, 16

`size`
 `SLinkedList< T >`, 16

`SLinkedList`
 `SLinkedList< T >`, 8, 9
`SLinkedList< T >`, 7
 `~SLinkedList`, 9
 `addFirst`, 9
 `addLast`, 10
 `clear`, 10
 `empty`, 10
 `insert`, 11
 `operator=`, 11, 12
 `operator==`, 12
 `operator[]`, 13
 `peek`, 13, 14
 `pop`, 14, 15
 `remove`, 16
 `size`, 16
 `SLinkedList`, 8, 9
`SLinkedList.hpp`, 17
 `operator<<`, 18