C Data structures

1.0

Generated by Doxygen 1.8.3.1

Mon Oct 7 2013

Contents

1 File Index **3**

1.1 File List . . . 3

2 File Documentation **5**

2.1 BinaryTree.h File Reference . . . 5

2.1.1 Detailed Description . . . 6

2.1.2 Function Documentation . . . 6

2.1.2.1 sBSTContains . . . 6

2.1.2.2 sBSTDelete . . . 6

2.1.2.3 sBSTDestroy . . . 6

2.1.2.4 sBSTInit . . . 6

2.1.2.5 sBSTInsert . . . 7

2.1.2.6 sBSTSearch . . . 7

2.1.2.7 sBSTSize . . . 7

2.1.2.8 sBSTTraverse . . . 7

2.2 BinomialHeap.h File Reference . . . 8

2.2.1 Detailed Description . . . 8

2.2.2 Function Documentation . . . 9

2.2.2.1 sBinHeapChangeKey . . . 9

2.2.2.2 sBinHeapDelete . . . 9

2.2.2.3 sBinHeapDeleteExtreme . . . 9

2.2.2.4 sBinHeapDestroy . . . 9

2.2.2.5 sBinHeapFindExtreme . . . 9

2.2.2.6 sBinHeapInit . . . 10

2.2.2.7 sBinHeapInsert . . . 10

2.2.2.8 sBinHeapMerge . . . 10

2.3 DynamicArray.h File Reference . . . 11

2.3.1 Detailed Description . . . 11

2.3.2 Function Documentation . . . 11

2.3.2.1 dArrayAdd . . . 11

2.3.2.2 dArrayClear . . . 12

2.3.2.3 dArrayDestroy . . . 12
2.3.2.4 dArrayErase . . . 12
2.3.2.5 dArrayGet . . . 12
2.3.2.6 dArrayInit . . . 12
2.3.2.7 dArrayResize . . . 13
2.3.2.8 dArraySet . . . 13
2.3.2.9 dArraySize . . . 13
2.4 LinkedList.h File Reference . . . 13
2.4.1 Detailed Description . . . 14
2.4.2 Typedef Documentation . . . 14
2.4.2.1 sLinkedList . . . 14
2.4.2.2 sListIterator . . . 15
2.4.3 Function Documentation . . . 15
2.4.3.1 listClear . . . 15
2.4.3.2 listDestroy . . . 15
2.4.3.3 listEmpty . . . 15
2.4.3.4 listErase . . . 15
2.4.3.5 listGet . . . 15
2.4.3.6 listHead . . . 16
2.4.3.7 listInitialize . . . 16
2.4.3.8 listInsert . . . 16
2.4.3.9 listIteratorCopy . . . 17
2.4.3.10 listIteratorDestroy . . . 17
2.4.3.11 listIteratorEnd . . . 17
2.4.3.12 listIteratorNext . . . 17
2.4.3.13 listPeek . . . 17
2.4.3.14 listPopBack . . . 18
2.4.3.15 listPopFront . . . 18
2.4.3.16 listPushBack . . . 18
2.4.3.17 listPushFront . . . 19
2.4.3.18 listSize . . . 19
2.5 Stack.h File Reference . . . 19
2.5.1 Detailed Description . . . 20
2.5.2 Function Documentation . . . 20
2.5.2.1 sStackDestroy . . . 20
2.5.2.2 sStackInit . . . 20
2.5.2.3 sStackPeek . . . 20
2.5.2.4 sStackPop . . . 20
2.5.2.5 sStackPush . . . 21
2.5.2.6 sStackSize . . . 21

Index 21

Chapter 1

File Index

1.1 File List

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

BinaryTree.h
A Generic binary tree implementation 5
BinomialHeap.h
A Generic binomial heap implementation. Requires Stack.h 8
DynamicArray.h
A Generic Dynamic Array implementation 11
LinkedList.h
A Generic Linked List 13
Stack.h
A Generic Stack implementation 19

Chapter 2

File Documentation

2.1 BinaryTree.h File Reference

A Generic binary tree implementation.

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

Macros

- #define **BSTTRAVERSE_INORDER** (char)0
- #define **BSTTRAVERSE_PREORDER** (char)1
- #define **BSTTRAVERSE_POSTORDER** (char)2

Typedefs

- typedef struct sBSTree **sBSTree**

Functions

- int sBSTInit (sBSTree ∗∗Tree, size_t ElementSize, void(∗EraseFun)(void ∗))

  Initialize a Binary Search Tree instance.
- void sBSTInsert (sBSTree ∗Tree, long int Key, void ∗Data, int Replace)

  Insert a copy of data into the tree.
- void ∗ sBSTSearch (sBSTree ∗Tree, long int Key)

  Search for a given key in the tree.
- int sBSTContains (sBSTree ∗Tree, long int Key)

  Checks if the tree contains a given key.
- void sBSTDelete (sBSTree ∗Tree, long int Key)

  Delete a node from the tree, given its key.
- void ∗∗ sBSTTraverse (sBSTree ∗Tree, char Order, size_t ∗Size)

  Traverses the tree based on a given order.
- void sBSTDestroy (sBSTree ∗∗Tree)

  Delete every node in the given tree as well as the tree itself. Deinitializes all initialized data.
- size_t sBSTSize (sBSTree ∗Tree)

  Returns the number of nodes in a given tree.

2.1.1 Detailed Description

A Generic binary tree implementation. Jimmy Holm

Date

October 7, 2013

2.1.2 Function Documentation

2.1.2.1 int sBSTContains (sBSTree * *Tree,* long int *Key*)

Checks if the tree contains a given key.

Parameters

Tree	a pointer to the tree instance to search in
Key	the key to be searched for

Returns

1 if the key exists in the tree, 0 if it does not. Examines the tree, returning a boolean value regarding the existence of a given key.

2.1.2.2 void sBSTDelete (sBSTree * *Tree,* long int *Key*)

Delete a node from the tree, given its key.

Parameters

Tree	a pointer to the tree instance to delete from
Key	the key of the node to be deleted Deletes a node from the tree

2.1.2.3 void sBSTDestroy (sBSTree ** *Tree*)

Delete every node in the given tree as well as the tree itself. Deinitializes all initialized data.

Parameters

Tree	reference to the tree instance to be destroyed. Will delete the entire tree and release all used resources. Upon exiting, the provided Tree pointer will point to NULL.

2.1.2.4 int sBSTInit (sBSTree ** *Tree,* size_t *ElementSize,* void(*)(void *) *EraseFun*)

Initialize a Binary Search Tree instance.

Parameters

Tree	a reference to the tree instance to be initialized.
ElementSize	size in bytes of the data stored by the tree.
EraseFun	pointer to a function used to deinitialize stored elements prior to deletion.

Returns

1 upon successful initialization, 0 otherwise. At the end of a successful function call, Tree will contain an initialized sBSTree instance.

2.1.2.5 void sBSTInsert (sBSTree * *Tree,* long int *Key,* void * *Data,* int *Replace*)

Insert a copy of data into the tree.

Parameters

Tree	a pointer to the tree instance to insert into.
Data	the data to be copied into the tree. Inserts a copy of the given data into the tree. If the key exists and Replace is non-null, the data of the existing node will be replaced with a copy of the data given.

2.1.2.6 void* sBSTSearch (sBSTree * *Tree,* long int *Key*)

Search for a given key in the tree.

Parameters

Tree	a pointer to the tree instance to search in
Key	the key to be searched for.

Returns

Returns a pointer to the stored data if found, or NULL if not found. Searches the tree for a given key, returning the data pointer.

2.1.2.7 size_t sBSTSize (sBSTree * *Tree*)

Returns the number of nodes in a given tree.

Parameters

Tree	pointer to an initialized tree instance

Returns

The number of nodes in the given tree Returns the number of nodes in a given tree

2.1.2.8 void** sBSTTraverse (sBSTree * *Tree,* char *Order,* size_t * *Size*)

Traverses the tree based on a given order.

Parameters

Tree	a pointer to the tree instance to traverse
Order	the traversing order to use

Remarks

Order can be one of BSTTRAVERSE_INORDER, BSTTRAVERSE_PREORDER or BSTTRAVERSE_POSTORDER.

Parameters

Size	if not null, this will contain the number of elements in the array.

Returns

An n-sized array, where n is the number of nodes in the tree, containing every node's data, sorted by traversation order. Traverses the tree based on a given order, returning an array containing each node's data.

2.2 BinomialHeap.h File Reference

A Generic binomial heap implementation. Requires Stack.h.

Macros

- #define **BINOM_MINHEAP** (char)0
- #define **BINOM_MAXHEAP** (char)1

Typedefs

- typedef struct sBinomHeap **sBinomHeap**

Functions

- int sBinHeapInit (sBinomHeap **Heap, size_t ElementSize, void(*EraseFun)(void *), char HeapType)
  Initialize a binomial heap.
- void sBinHeapMerge (sBinomHeap **Heap1, sBinomHeap **Heap2)
  Merge two heaps.
- void sBinHeapInsert (sBinomHeap *Heap, long Key, void *Data)
  Copy data into the heap.
- void * sBinHeapFindExtreme (sBinomHeap *Heap)
  Find the node with the lowest/highest key value in the heap, depending on the type of heap.
- int sBinHeapChangeKey (sBinomHeap *Heap, long Key, long NewKey)
  Change the key value of a given node, while retaining the min/max-heap property.
- void sBinHeapDeleteExtreme (sBinomHeap *Heap)
  Delete the node with the maximum/minimum key value depending on the heap type.
- void sBinHeapDelete (sBinomHeap *Heap, long Key)
  Delete a node with the given key value.
- void sBinHeapDestroy (sBinomHeap **Heap)
  Destroy a given heap.

2.2.1 Detailed Description

A Generic binomial heap implementation. Requires Stack.h. Jimmy Holm

Date

October 7, 2013

2.2.2 Function Documentation

2.2.2.1 int sBinHeapChangeKey (sBinomHeap * *Heap,* long *Key,* long *NewKey*)

Change the key value of a given node, while retaining the min/max-heap property.

Parameters

Heap	a binomial heap instance pointer to the heap containing the element we wish to decrease
Key	the key of the node to change
NewKey	the key of the node to change to. For a min-heap, the new key must be less than Key - for a max-heap the new key must be greater.

Returns

Returns 1 if the given node existed, and had its key changed - 0 otherwise. Alters the key value of a given node while retaining the heap properties.

2.2.2.2 void sBinHeapDelete (sBinomHeap * *Heap,* long *Key*)

Delete a node with the given key value.

Parameters

Heap	a binomial heap instance pointer to the heap that will have a given node deleted.
Key	the key value of the node to be deleted. Deletes a node with the given key value from the heap. Calls upon the heap's EraseFun if available.

2.2.2.3 void sBinHeapDeleteExtreme (sBinomHeap * *Heap*)

Delete the node with the maximum/minimum key value depending on the heap type.

Parameters

Heap	a binomial heap instance pointer to the heap that will have its extreme node deleted. Deletes the node with the extreme key value depending on the heap type. Calls upon the heap's EraseFun if available.

2.2.2.4 void sBinHeapDestroy (sBinomHeap ** *Heap*)

Destroy a given heap.

Parameters

Heap	reference to a binomial heap instance pointer to the heap to be destroyed. Deletes every node in the tree, releasing all kept resources and calling the EraseFun, if available, on each element in turn. Upon finishing, the given Heap pointer will point to NULL.

2.2.2.5 void* sBinHeapFindExtreme (sBinomHeap * *Heap*)

Find the node with the lowest/highest key value in the heap, depending on the type of heap.

Parameters

Heap	a binomial heap instance pointer to search in

Returns

The data held by the node with the lowest key value. Searches through the heap for the node with the lowest key value.

2.2.2.6 int sBinHeapInit (sBinomHeap ** *Heap*, size_t *ElementSize*, void(*)(void *) *EraseFun*, char *HeapType*)

Initialize a binomial heap.

Parameters

Heap	a reference to a binomial heap instance pointer to be initialized, should be assigned 0 before this function is called.
ElementSize	the size of a stored element
EraseFun	pointer to an optional erasure function in charge of deinitializing an element before it's erased from the heap.
HeapType	determines whether the heap is to be treated as a min heap or max heap, using the macros BINOM_MINHEAP and BINOM_MAXHEAP

Returns

1 if initialization is successful, 0 otherwise. Initializes a NULL assigned binomial heap pointer and preparing it for use.

2.2.2.7 void sBinHeapInsert (sBinomHeap * *Heap*, long *Key*, void * *Data*)

Copy data into the heap.

Parameters

Heap	a binomial heap instance pointer to insert a copy of the given data into.
Key	the key value to be given to the new data node
Data	the data value to be copied into the heap Creates a copy of the given data and inserts it into the heap. Note that a of the data is added and managed by the heap; the lifetime of the original data is managed.

2.2.2.8 void sBinHeapMerge (sBinomHeap ** *Heap1*, sBinomHeap ** *Heap2*)

Merge two heaps.

Parameters

Heap1	a reference to a binomial heap instance pointer to the first of the two heaps to be merged.
Heap2	a reference to a binomial heap instance pointer to the second of the two heaps to be merged. Merges two heaps into a new heap. Upon successful completion of the function, Heap1 will contain the merged heap and Heap 2 will be NULL.

2.3 DynamicArray.h File Reference

A Generic Dynamic Array implementation.

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

Typedefs

- typedef struct sDynamicArray **sDynamicArray**

Functions

- int dArrayInit (sDynamicArray **Array, size_t Size, size_t ElementSize, void(*EraseFun)(void *))
  Initialize an Array object.
- void dArrayDestroy (sDynamicArray **Array)
  Destroy a previously initialized array.
- void dArraySet (sDynamicArray *Array, size_t Index, void *Data, int Erase)
  Set the value stored at a given index in the array.
- void * dArrayGet (sDynamicArray *Array, size_t Index)
  Get the data stored at a given index in the array.
- void dArrayErase (sDynamicArray *Array, size_t Index)
  Erase the data stored at a given index in the array.
- void dArrayClear (sDynamicArray *Array)
  Erase all elements in the array, without resizing it.
- int dArrayResize (sDynamicArray *Array, size_t NewSize)
  Resize the array to fit more or fewer elements.
- int dArraySize (sDynamicArray *Array)
  Return the maximum number of elements the given array fits.
- int dArrayAdd (sDynamicArray *Array, void *Data, int Erase)
  Add data to the first available element slot, resizing if necessary.

2.3.1 Detailed Description

A Generic Dynamic Array implementation. Jimmy Holm

Date

October 7, 2013

2.3.2 Function Documentation

2.3.2.1 int dArrayAdd (sDynamicArray * *Array*, void * *Data*, int *Erase*)

Add data to the first available element slot, resizing if necessary.

Parameters

Array	pointer to an initialized array instance
Data	void pointer to the data to be copied.
Erase	boolean value to determine whether the data already stored should be erased and freed.

Returns

Returns the index of the newly added element. dArrayAdd adds a copy of the given data to the array, resizing the array if necessary.

2.3.2.2 void dArrayClear (sDynamicArray * *Array*)

Erase all elements in the array, without resizing it.

Parameters

Array	to be cleared. dArrayClear erases all the elements stored in the array without resizing the array.

2.3.2.3 void dArrayDestroy (sDynamicArray ** *Array*)

Destroy a previously initialized array.

Parameters

Array	reference to a previously initialized Array instance Erase all valid elements in the array and reset the Array instance pointer

2.3.2.4 void dArrayErase (sDynamicArray * *Array,* size_t *Index*)

Erase the data stored at a given index in the array.

Parameters

Array	pointer to an initialized array instance.
Index	the index of the array element to erase dArrayErase erases the data stored at a given index

2.3.2.5 void* dArrayGet (sDynamicArray * *Array,* size_t *Index*)

Get the data stored at a given index in the array.

Parameters

Array	pointer to an initialized array instance.
Index	index into the array to be update. dArrayGet returns the data stored at a given index.

2.3.2.6 int dArrayInit (sDynamicArray ** *Array,* size_t *Size,* size_t *ElementSize,* void(*)(void *) *EraseFun*)

Initialize an Array object.

Parameters

Array	reference to the Array instance to be initialized. The Array instance should be set to NULL before being passed to the function.
Size	the number of elements to allocate for at initialization
ElementSize	the size, in bytes, of a single stored element
EraseFun	a function pointer to a function used to deinitialize a stored object before it's freed.

Returns

1 upon successfully creating and initializing, 0 otherwise Creates a useable Array object of a given size.

2.3.2.7 int dArrayResize (sDynamicArray * *Array,* size_t *NewSize*)

Resize the array to fit more or fewer elements.

Parameters

Array	pointer to an initialized array instance
NewSize	the size, in maximum number of elements, of the array. dArrayResize returns 1 on successful resize, 0 otherwise.

2.3.2.8 void dArraySet (sDynamicArray * *Array,* size_t *Index,* void * *Data,* int *Erase*)

Set the value stored at a given index in the array.

Parameters

Array	pointer to an initialized array instance.
Index	index into the array to be updated.
Data	void pointer to the data to be copied into the array.
Erase	boolean value to determine whether the data already stored should be erased and freed. dArraySet sets the stored value of an array element to a new value Data. If Erase is set to 1, the data already stored will be freed.

2.3.2.9 int dArraySize (sDynamicArray * *Array*)

Return the maximum number of elements the given array fits.

Parameters

Array	pointer to an initialized array instance

Returns

The maximum number of elements the given array can store. dArraySize returns the maximum number of elements the given array fits.

2.4 LinkedList.h File Reference

A Generic Linked List.

```
#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 * listPeek (sLinkedList *List)

  Return the data of the first element of the list.
- 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 listDestroy (sLinkedList **List)

  Destroy the list.
- void listIteratorNext (sListIterator *Iterator)

  Advance an iterator to the next element in a list.
- int listIteratorEnd (sListIterator *Iterator)

  Check whether or not an iterator is at the end of its list.
- void listIteratorCopy (sListIterator *Src, sListIterator **Dst)

  Creates a copy of a given iterator.
- void listIteratorDestroy (sListIterator **Iterator)

  Destroy an iterator.

2.4.1 Detailed Description

A Generic Linked List. Jimmy Holm, Marcus Münger

Date

September 25, 2013

2.4.2 Typedef Documentation

2.4.2.1 typedef struct sLinkedList sLinkedList

Linked List structure

2.4.2.2 typedef struct sListIterator sListIterator

Generic Iterator for linked lists

2.4.3 Function Documentation

2.4.3.1 void listClear (sLinkedList * *List*)

Clear the list.

Parameters

List	a pointer to an initialized list. listClear calls listErase on every element in the list, resulting in an empty list

2.4.3.2 void listDestroy (sLinkedList ** *List*)

Destroy the list.

Parameters

List	a reference to an initialized list pointer listDestroy destroyes and performs cleanup on a list, empty or otherwise.

2.4.3.3 int listEmpty (sLinkedList * *List*)

Check whether the list is empty or contains elements.

Parameters

List	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.

2.4.3.4 void listErase (sListIterator * *Iterator*)

Erase the element pointed to by Iterator.

Parameters

Iterator	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()

2.4.3.5 void* listGet (sListIterator * *Iterator*)

Return the data held by an iterator.

Parameters

Iterator	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()

2.4.3.6 void listHead (sLinkedList * *List*, sListIterator ** *It*)

Initialize an iterator to the head of the list.

Parameters

List	a pointer to an initialized list.
It	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.

2.4.3.7 void listInitialize (sLinkedList ** *List*, size_t *ElementSize*, void(*)(void *) *EraseFun*)

Initialize a linked list.

Parameters

List	a reference to an uninitialized list pointer.
ElementSize	the size of a list's stored data.
EraseFun	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 deconstructor 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.

2.4.3.8 void listInsert (sListIterator * *Iterator*, void * *Data*)

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

Parameters

Iterator	pointer to an initialized iterator into a list, where the new element is to be inserted.
Data	a pointer to the data to be copied into the list. This function inserts a copy of the provided data into the list in front of the current iterator position. 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.

2.4.3.9 void listIteratorCopy (sListIterator * *Src*, sListIterator ** *Dst*)

Creates a copy of a given iterator.

Parameters

Src	an initialized iterator into an initialized list.
Dst	an initialized or NULL-pointing iterator to be set. listIteratorCopy sets a destination iterator to point to the same list element as the given source iterator.

2.4.3.10 void listIteratorDestroy (sListIterator ** *Iterator*)

Destroy an iterator.

Parameters

Iterator	reference to an initialized iterator to be destroyed listIteratorDestroy destroyes the given iterator reference.

2.4.3.11 int listIteratorEnd (sListIterator * *Iterator*)

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

Parameters

Iterator	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.

2.4.3.12 void listIteratorNext (sListIterator * *Iterator*)

Advance an iterator to the next element in a list.

Parameters

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

2.4.3.13 void* listPeek (sLinkedList * *List*)

Return the data of the first element of the list.

Parameters

List	pointer to an initialized list

Returns

the data held by the list's head or NULL for an empty list listPeek returns the data stored in the list's head element.

2.4.3.14 void listPopBack (sLinkedList * *List*)

Pop the last element of the list.

Parameters

List	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()

2.4.3.15 void listPopFront (sLinkedList * *List*)

Pop the first element of the list.

Parameters

List	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()

2.4.3.16 void listPushBack (sLinkedList * *List,* void * *Data*)

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

Parameters

List	a pointer to a list previously initialized with listInitialized, into which Data is to be added.
Data	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()

2.4.3.17 void listPushFront (sLinkedList * *List*, void * *Data*)

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

Parameters

List	a pointer to a list previously initialized with listInitialized, into which Data is to be added.
Data	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 *copy* 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()

2.4.3.18 size_t listSize (sLinkedList * *List*)

Return the number of elements in a list.

Parameters

List	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.

2.5 Stack.h File Reference

A Generic Stack implementation.

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

Typedefs

- typedef struct sStack **sStack**

Functions

- int sStackInit (sStack **Stack, size_t ElementSize, void(*EraseFun)(void *))

  Initialize a stack pointer.
- void sStackPush (sStack *Stack, void *Data)

  Copies data onto the stack.
- void * sStackPeek (sStack *Stack)

  Retrieve the data on the top of the stack.
- void sStackPop (sStack *Stack)

  Remove the top-most element on the stack.
- size_t sStackSize (sStack *Stack)

  Return the number of elements on the stack.
- void sStackDestroy (sStack **Stack)

  Destroy a stack pointer.

2.5.1 Detailed Description

A Generic Stack implementation. Jimmy Holm

Date

October 7, 2013

2.5.2 Function Documentation

2.5.2.1 void sStackDestroy (sStack ∗∗ *Stack*)

Destroy a stack pointer.

Parameters

Stack	a reference to the stack pointer to be destroyed. Destroy a stack pointer, releasing all its stored resources and resetting the pointer to NULL.

2.5.2.2 int sStackInit (sStack ∗∗ *Stack,* size_t *ElementSize,* void(∗)(void ∗) *EraseFun*)

Initialize a stack pointer.

Parameters

Stack	a reference to the stack pointer to be initialized. Must be NULL.
ElementSize	size of the data stored.
EraseFun	function used to deinitialized stored data.

Returns

Returns 1 upon successfully initiating Stack, 0 otherwise. Initializes a stack pointer, preparing it for use.

2.5.2.3 void∗ sStackPeek (sStack ∗ *Stack*)

Retrieve the data on the top of the stack.

Parameters

Stack	the stack pointer to peek into

Returns

The data held by the top of the stack Returns the data held by the top of the stack.

2.5.2.4 void sStackPop (sStack ∗ *Stack*)

Remove the top-most element on the stack.

Parameters

Stack	the stack which is to have its top element removed. The topmost element of Stack is removed, with EraseFun called on it to deinitialize prior to having its resources released.

2.5.2.5 void sStackPush (sStack * *Stack*, void * *Data*)

Copies data onto the stack.

Parameters

Stack	the stack to have data pushed onto.
Data	the data to be copied onto the stack. Pushes a copy of Data onto the top of the stack.

2.5.2.6 size_t sStackSize (sStack * *Stack*)

Return the number of elements on the stack.

Parameters

Stack	the stack which size is requested. Returns the number of elements stored on the given stack.

Index

BinaryTree.h, 5
    sBSTContains, 6
    sBSTDelete, 6
    sBSTDestroy, 6
    sBSTInit, 6
    sBSTInsert, 7
    sBSTSearch, 7
    sBSTSize, 7
    sBSTTraverse, 7
BinomialHeap.h, 8
    sBinHeapChangeKey, 9
    sBinHeapDelete, 9
    sBinHeapDeleteExtreme, 9
    sBinHeapDestroy, 9
    sBinHeapFindExtreme, 9
    sBinHeapInit, 10
    sBinHeapInsert, 10
    sBinHeapMerge, 10

dArrayAdd
    DynamicArray.h, 11
dArrayClear
    DynamicArray.h, 12
dArrayDestroy
    DynamicArray.h, 12
dArrayErase
    DynamicArray.h, 12
dArrayGet
    DynamicArray.h, 12
dArrayInit
    DynamicArray.h, 12
dArrayResize
    DynamicArray.h, 13
dArraySet
    DynamicArray.h, 13
dArraySize
    DynamicArray.h, 13
DynamicArray.h, 11
    dArrayAdd, 11
    dArrayClear, 12
    dArrayDestroy, 12
    dArrayErase, 12
    dArrayGet, 12
    dArrayInit, 12
    dArrayResize, 13
    dArraySet, 13
    dArraySize, 13

LinkedList.h, 13
    listClear, 15
    listDestroy, 15
    listEmpty, 15
    listErase, 15
    listGet, 15
    listHead, 16
    listInitialize, 16
    listInsert, 16
    listIteratorCopy, 17
    listIteratorDestroy, 17
    listIteratorEnd, 17
    listIteratorNext, 17
    listPeek, 17
    listPopBack, 18
    listPopFront, 18
    listPushBack, 18
    listPushFront, 18
    listSize, 19
    sLinkedList, 14
    sListIterator, 14
listClear
    LinkedList.h, 15
listDestroy
    LinkedList.h, 15
listEmpty
    LinkedList.h, 15
listErase
    LinkedList.h, 15
listGet
    LinkedList.h, 15
listHead
    LinkedList.h, 16
listInitialize
    LinkedList.h, 16
listInsert
    LinkedList.h, 16
listIteratorCopy
    LinkedList.h, 17
listIteratorDestroy
    LinkedList.h, 17
listIteratorEnd
    LinkedList.h, 17
listIteratorNext
    LinkedList.h, 17
listPeek
    LinkedList.h, 17
listPopBack
    LinkedList.h, 18
listPopFront
    LinkedList.h, 18

listPushBack
LinkedList.h, 18
listPushFront
LinkedList.h, 18
listSize
LinkedList.h, 19

sBSTContains
BinaryTree.h, 6
sBSTDelete
BinaryTree.h, 6
sBSTDestroy
BinaryTree.h, 6
sBSTInit
BinaryTree.h, 6
sBSTInsert
BinaryTree.h, 7
sBSTSearch
BinaryTree.h, 7
sBSTSize
BinaryTree.h, 7
sBSTTraverse
BinaryTree.h, 7
sBinHeapChangeKey
BinomialHeap.h, 9
sBinHeapDelete
BinomialHeap.h, 9
sBinHeapDeleteExtreme
BinomialHeap.h, 9
sBinHeapDestroy
BinomialHeap.h, 9
sBinHeapFindExtreme
BinomialHeap.h, 9
sBinHeapInit
BinomialHeap.h, 10
sBinHeapInsert
BinomialHeap.h, 10
sBinHeapMerge
BinomialHeap.h, 10
sLinkedList
LinkedList.h, 14
sListIterator
LinkedList.h, 14
sStackDestroy
Stack.h, 20
sStackInit
Stack.h, 20
sStackPeek
Stack.h, 20
sStackPop
Stack.h, 20
sStackPush
Stack.h, 20
sStackSize
Stack.h, 21
Stack.h, 19
sStackDestroy, 20
sStackInit, 20
sStackPeek, 20
sStackPop, 20
sStackPush, 20
sStackSize, 21