RM Linked List

v1.2

Generated by Doxygen 1.8.11

Contents

Chapter 1

Singly Linked Lists:

A versatile library containing many functions that allow you to use and manipulate singly linked lists. The code was written with certain abstractions in mind, that allow easy modification of the to adapt the library's functionality

> *Note: The Singly Linked Lists library used to be part of a collection of Algorithms and Data Structures (AnD Pack) that I was working on, until I decided to split it off into a separate repository*

Documentation:

To generate the Documentation, just run the `doxygen` command in the root directory of this porject

Implementation:

- C Language following the C-11 Standard
- CMake v3.6 for compiling and organizing the build process
- For Documentation Generation: Doxygen v1.8.11

Naming Conventions:

All labels are suffixed with **ds_sll_**

example function: **ds_sll_createNode(∗...∗)**

example typedef: **ds_sll_node_t**

Data Types:

- Linked List Header: **ds_sll_t**
- Node: **ds_sll_node_t**
- Error Codes: **ds_sll_error_t**
- Executable Function Return Type: **ds_sll_func_return_t**

Overview:

The linked list is represented by a header, that keeps track of the **head** and the **tail**. The linked list consists of **nodes**. Each node has a reference to the **next node**. Each node also stores one **void pointer** that we call an **element**. This pointer is generic and can be used anyway you see fit. The simplest example is it could be a pointer to a primitive data type allocated in memory, and when you access this data you simply typecast the pointer to the appropriate type and then dereference it as you would any normal pointer. For the more advanced applications out there this pointer could reference more complicated objects or be used for more complex representations. To accommodate for the wide possiblities I aimed to abstract the library to manageable and easily customizable chunks.

Available Functions:

Create/Delete

- **ds_sll_newSinglyLinkedList**: Create a new header
- **ds_sll_createNode**: Create a new node
- **ds_sll_destroySinglyLinkedList**: Destroy a list and all its nodes
- **ds_sll_deleteNode**: Delete and free resources associated with a given Node
- **ds_sll_deleteNodeAtIndex**: Delete the node located at the given index, and free allocated resources.

Operations on Node:

- **ds_sll_nextNode**: Specifies how to get the next node following a given node
- **ds_sll_extractElementFromNode**: Extract an element stored in a node
- **ds_sll_storeElementInNode**: Store an element in a node
- **ds_sll_deleteElement**: Delete an element in a node and free its resources
- **ds_sll_copyElement**: Create a copy of a given element

Operations on List:

- **ds_sll_executeFunctionOnElements**: Execute a given function on all the nodes in the list
- **ds_sll_calculateLength**: Calculates the length of the linked list
- **ds_sll_splitSinglyLinkedListAtIndex**: Splits a linked list into two at the given index

Retrieval and Search:

- **ds_sll_getNodeAtIndex**: Retrieve the node at the given index
- **ds_sll_getElementAtIndex**: Retrieve the element in the node at the given index
- **ds_sll_findNodeContainingElement**: Searches the linked list for the node containing the given element

Append:

- **ds_sll_appendNode**: Append a node to the end of the list
- **ds_sll_appendElement**: Create a new node and store the given element in it and append the node to the end of the list
- **ds_sll_appendElementCopy**: Create a new node and store a copy of the given element in it and appen the node to the end of the list

Insert:

- **ds_sll_insertNodeAtIndex**: Insert a node at the given index
- **ds_sll_insertElementAtIndex**: Create a new node and store the given element and insert the new node at the given index in the list.
- **ds_sll_insertElementCopyAtIndex**: Create a new node and store a copy of the given element and insert the new node at the given index in the list.

Helper Functions:

- **ds_sll_traverseNodeToIndex**: A helper function that traverses a linked list and sets the given pointer to point to the node at the given index. It also returns an error code detailing what kind of error occurred.

executeFunctionOnElements:

This function allows you to execute a given function on the entire linked list in sequential order. This function takes an *executable function*, traverses the given linked list, and calls the given *executable function* on each node. This traversal is sequential and can be controlled via the return parameter of the *executable function*'s. The way to use is by defining a function in the form:

```
ds_sll_func_return_t myfunction(void* element, ds_sll_node_t* node, int
      index, void* shared) {
    ds_sll_func_return_t execution_status;

    ...

    rerturn execution_status;
}
```

Where obviously you can name the function anything you like. The first parameter `void* element` is a pointer to the element within the current node that `ds_sll_executeFunctionOnElements(...)` is processing. The second parameter `ds_sll_node_t* node` is a pointer to the node the function was called on The third parameter `int index` is the index of the node that this function was called on. The fourth parameter `void* shared` is used for three reasons:

- to allow your function to return some value back to you
- to share information between subsequent function calls
- to pass an initial input value to your functions The drawback is that you can only have one value to use for input, data sharing, and output. The return value of the function is used to control the execution flow, meaning it tells the `ds_sll_executeFunctionOnElements(...)` function whether:
- the function call to `myfunction(...)` was successful: continue to execute `myfunction(...)` on next node
- the function call to `myfunction(...)` ran into an error: do not continue executing the function on the next node, and return an appropriate error value back to the caller of `ds_sll_executeFunctionOn↩Elements(...)`
- the function ran successfully and would like to terminate: Used when your function is looking for a specific element and once it is found, there is no point in continuing to traverse the linked list therefore your function should request to terminate execution.

An example usage of `ds_sll_executeFunctionOnElements(...)` is a function that finds the maximum element in the linked list. The functions definitions is:

```
/* Stores the maximum int in `maxElement` */
ds_sll_func_return_t getMaxElement(void* element,
      ds_sll_node_t* node, int index, void* maxElement) {
    if(element == NULL) {
        return DS_SLL_CONTINUE_EXECUTION;
    }
    else if(maxElement == NULL) {
        *(int*)maxElement = *(int*)element;
    } else if(*(int*)element > *(int*)maxElement) {
        *(int*)maxElement = *(int*)element;
    }
    return DS_SLL_CONTINUE_EXECUTION;
}
```

```
int *max_element = (int*)malloc(sizeof(int));
ds_sll_executeFunctionOnElements(mylist, getMaxElement, (void*)max_element)
      ;
printf("Max Element in list: %d\n", *max_element);
free(max_element);
```

Another example is a function that returns a pointer to the node containing the max element:

```
ds_sll_func_return_t maxNode(void* element, ds_sll_node_t* node, int index
      , void* maxNode) {
    // maxNode is a pointer to a pointer to a ds_sll_node_t
    ds_sll_node_t **maximum_node_p = (ds_sll_node_t**)maxNode;
    if(element == NULL) {
        return DS_SLL_CONTINUE_EXECUTION;
    }
    else if(*maximum_node_p == NULL) {
        *maximum_node_p = node;
    } else if(*(int*)element > *(int*)ds_sll_extractElementFromNode(*
      maximum_node_p)) {
        *(ds_sll_node_t**)maxNode = node;
    }
    return DS_SLL_CONTINUE_EXECUTION;
}
```

```
ds_sll_node_t* max_node = NULL;
ds_sll_executeFunctionOnElements(mylist, maxNode, (void*)&max_node);
int* max_node_element = (int*)ds_sll_extractElementFromNode(max_node);
printf("Max Element in Max Node is: %d\n", *max_node_element);
```

A final example shows how you can define your own data structure to allow you to pass in multiple parameters to your function. This function will stop traversing the node when the element you were looking for is found by returning `DS_SLL_STOP_EXECUTION`:

```
typedef struct findNodeFuncParams {
    int input;
    int output_index;
    ds_sll_node_t* output_node;
} findNodeFuncParams;
```

```
ds_sll_func_return_t findNode(void* element, ds_sll_node_t* node, int
      index, void* shared) {
    findNodeFuncParams* params = (findNodeFuncParams*)shared;

    if(params->input == *(int*)element) {
        params->output_index = index;
        params->output_node = node;
        return DS_SLL_STOP_EXECUTION;
    } else {
        return DS_SLL_CONTINUE_EXECUTION;
    }
}
```

```
findNodeFuncParams* params = (findNodeFuncParams*) malloc(sizeof(findNodeFuncParams));
params->input = 77;
ds_sll_executeFunctionOnElements(mylist, findNode, (void*)params);
printf("Node containing element (%d) is located at index %d\n", params->input, params->output_index);
free(params);
```

Please note that if you alter the element entry in the `ds_sll_node_t` *data structure to be anything other than a* `void*` *you might have to edit one or more other functions, mainly you will need to alter the* `ds_sll_executeFunctionOnElements(...)` *function as it passes a* `void*` *as an argument to the given executable function. This does not mean that you can not have a complex data type for the* ***element*** *of your node, this library was designed so that no matter what the data type is of your element the node only stores a generic void pointer to refer to it, and the library functions depend on the usage of* `ds_sll_extractElementFromNode`, `ds_sll_storeElementInNode`, `ds_sll_deleteElement`, `ds_sll_copyElement`, *and other abstractions to appropriately access and manipulate the element objects.*

So as a general guideline, to easily customize the library, stick to modifying the library functions, mainly those mentioned below in the ***Abstractions*** *section.*

Abstractions:

This library was designed with some abstractions in mind to allow for easy customization according to your project's needs. These abstractions are mainly implemented as inline functions where possible. The main abstractions are:

- **ds_sll_createNode**: allows you to specify how nodes are created
- **ds_sll_extractElementFromNode**: allows you to specify how node elements are read (return node->element)
- **ds_sll_storeElementInNode**: allows you to specify how to insert an element in a node (node->element = new_element)
- **ds_sll_deleteElement**: allows you to specify how to deallocate the resources of a given Element
- **ds_sll_deleteNode**: allows you to specify how a node is 'deleted' and its resources are de-allocated
- **ds_sll_copyElement**: allows you to specify how an Element is copied

You can alter these functions and define your own custom functions using `ds_sll_executeFunctionOnElements` to suite the needs of your application and you can easily change the way data is being stored in the node. Maybe the void pointers being stored in the nodes are references to more complicated objects, and therefore you can modify these functions to perform initializations and data manipulations according to your design without affecting the overall linked list functionality.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

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

Chapter 5

Module Documentation

5.1 Singly Linked List

Files

- file SinglyLinkedList.c

  Singly Linked List library (ds_sll)
- file SinglyLinkedList.h

  Singly Linked List library (Header) (ds_sll)

Data Structures

- struct ds_sll_node_t
- struct ds_sll_t

Typedefs

- typedef struct ds_sll_node_t ds_sll_node_t
- typedef struct ds_sll_t ds_sll_t
- typedef enum ds_sll_error_t ds_sll_error_t
- typedef enum ds_sll_func_return_t ds_sll_func_return_t

Enumerations

- enum ds_sll_error_t {
  DS_SLL_NO_ERROR = 0, DS_SLL_NODE_CREATION_ERROR, DS_SLL_ELEMENT_CREATION_ER↩
  ROR, DS_SLL_LIST_CREATION_ERROR,
  DS_SLL_INDEX_OUT_OF_BOUNDS_ERROR, DS_SLL_BROKEN_LIST_ERROR, DS_SLL_LIST_TOO↩
  _SMALL_ERROR, DS_SLL_FUNCTION_EXECUTION_ERROR }
- enum ds_sll_func_return_t { DS_SLL_CONTINUE_EXECUTION, DS_SLL_STOP_EXECUTION, DS_SLL↩
  _EXECUTION_ERROR }

Functions

- ds_sll_t * ds_sll_newSinglyLinkedList ()

  Create a new singly linked list.
- ds_sll_node_t * ds_sll_createNode (void *element)

  Create a Node to be used with a singly linked list.
- ds_sll_error_t ds_sll_destroySinglyLinkedList (ds_sll_t *linkedList)

  Destroy a Singly Linked List.
- void ds_sll_deleteNode (ds_sll_node_t *node)

  Delete the given node; Free all allocated resources.
- ds_sll_error_t ds_sll_deleteNodeAtIndex (ds_sll_t *linkedList, int index)

  Delete the node at the given index from the singly linked list.
- ds_sll_node_t * ds_sll_nextNode (ds_sll_node_t *node)

  Inline function that returns the next node in the given list.
- void * ds_sll_extractElementFromNode (ds_sll_node_t *node)

  Extract the element contained in the given node.
- void ds_sll_storeElementInNode (ds_sll_node_t *node, void *element)

  Encapsulate an element within a given node.
- void ds_sll_deleteElement (void *node)

  Delete an element contained within a node and free any allocated resources.
- void * ds_sll_copyElement (void *element, const size_t element_size)

  Copy a given element to be stored in a node.
- int ds_sll_executeFunctionOnElements (ds_sll_t *linkedList, ds_sll_func_return_t(*func)(void *, ds_sll_node_t *, int, void *), void *sharedData)

  Executes a function on each element in the linked list in order.
- int ds_sll_calculateLength (const ds_sll_t *linkedList)

  Calculates the length of the singly linked list (by traversing it)
- ds_sll_error_t ds_sll_splitSinglyLinkedListAtIndex (ds_sll_t *firstLinkedList, ds_sll_t *secondLinkedList, int index)

  Split Singly Linked List into two singly linked lists at the given index.
- ds_sll_node_t * ds_sll_getNodeAtIndex (const ds_sll_t *linkedList, int index)

  Get the node from a singly linked list at a given index.
- void * ds_sll_getElementAtIndex (const ds_sll_t *linkedList, int index)

  Get the element contained in the node at the given index in the specified singly linked list.
- ds_sll_node_t * ds_sll_findNodeContainingElement (ds_sll_t *linkedList, void *element, int(*equalityFunc)(void *, void *), int *resultIndex)

  Find the node(s) containing the given element.
- void ds_sll_appendNode (ds_sll_t *linkedList, ds_sll_node_t *node)

  Append a given node to the end of the given singly linked list.
- ds_sll_error_t ds_sll_appendElement (ds_sll_t *linkedList, void *element)

  Create a new node encapsulating the given element, and append this node to the end of the given singly linked list.
- ds_sll_error_t ds_sll_appendElementCopy (ds_sll_t *linkedList, void *element, const size_t element_size)

  Create a new node encapsulating a copy of the given element, and append this node to the end of the given singly linked list.
- ds_sll_error_t ds_sll_insertNodeAtIndex (ds_sll_t *linkedList, ds_sll_node_t *node, int index)

  Inserts the given node at the chosen index.
- ds_sll_error_t ds_sll_insertElementAtIndex (ds_sll_t *linkedList, void *element, int index)

  Create a new node with the given element and inserts the new node at the chosen index.
- ds_sll_error_t ds_sll_insertElementCopyAtIndex (ds_sll_t *linkedList, void *element, const size_t element_size, int index)

  Create a new node with a copy of the given element and inserts the new node at the chosen index.
- ds_sll_error_t ds_sll_traverseNodeToIndex (const ds_sll_t *linkedList, ds_sll_node_t **node, int index)

  Taverse a linked list to the node at a given index.

5.1.1 Detailed Description

5.1.2 Typedef Documentation

5.1.2.1 typedef enum ds_sll_error_t ds_sll_error_t

Enum representation of return error codes

5.1.2.2 typedef enum ds_sll_func_return_t ds_sll_func_return_t

Singly Linked List Function return values. These are the return values that a function mapped to a singly linked list should return

5.1.2.3 typedef struct ds_sll_node_t ds_sll_node_t

Singly Linked List Node datatype

5.1.2.4 typedef struct ds_sll_t ds_sll_t

Singly Linked List datatype. This struct holds the information that identifies a Singly Linked List

5.1.3 Enumeration Type Documentation

5.1.3.1 enum ds_sll_error_t

Enum representation of return error codes

Enumerator

DS_SLL_NO_ERROR No Error

DS_SLL_NODE_CREATION_ERROR Error creating a new node

DS_SLL_ELEMENT_CREATION_ERROR Error allocating memory for an element

DS_SLL_LIST_CREATION_ERROR Error creating a new singly linked list

DS_SLL_INDEX_OUT_OF_BOUNDS_ERROR The index is out of bounds

DS_SLL_BROKEN_LIST_ERROR Error traversing a singly linked list till the end (the list is broken)

DS_SLL_LIST_TOO_SMALL_ERROR The length of given singly linked list is too small

DS_SLL_FUNCTION_EXECUTION_ERROR A function that was being executed on a Singly Linked List returned an Error

5.1.3.2 enum ds_sll_func_return_t

Singly Linked List Function return values. These are the return values that a function mapped to a singly linked list should return

Enumerator

DS_SLL_CONTINUE_EXECUTION Function ran successfully. Continue to next node

DS_SLL_STOP_EXECUTION Function has completed it's goal. Stop execution

DS_SLL_EXECUTION_ERROR Function encountered an Error

5.1.4 Function Documentation

5.1.4.1 ds_sll_error_t ds_sll_appendElement (ds_sll_t * *linkedList,* void * *element*)

Create a new node encapsulating the given element, and append this node to the end of the given singly linked list.

Parameters

linkedList	The singly linked list to append to
element	A pointer to the element to append to the linked list

Returns

1 if an error occurred; 0 otherwise

Possible Errors:

- Failure to create a new node

5.1.4.2 ds_sll_error_t ds_sll_appendElementCopy (ds_sll_t * *linkedList,* void * *element,* const size_t *element_size*)

Create a new node encapsulating a copy of the given element, and append this node to the end of the given singly linked list.

Parameters

linkedList	The singly linked list to append to
element	The element to copy and append the new copy to the end of the given list
element_size	The size in bytes of the given element

Returns

1 if an error occurred; 0 otherwise

5.1.4.3 void ds_sll_appendNode (ds_sll_t * *linkedList,* ds_sll_node_t * *node*)

Append a given node to the end of the given singly linked list.

Parameters

linkedList	The singly linked list to append the node to
node	The node to append to the end of the singly linked list

5.1.4.4 int ds_sll_calculateLength (const ds_sll_t * *linkedList*)

Calculates the length of the singly linked list (by traversing it)

Parameters

linkedList	The singly linked list that's length you seek

Returns

An Integer representing the length of the linked list, or a negative integer who's absolute value represents the index where an error occurred.

5.1.4.5 void* ds_sll_copyElement (void * *element*, const size_t *element_size*)

Copy a given element to be stored in a node.

Parameters

element	The element to copy
element_size	The size of the given element (in bytes) to be copied

Returns

The new copy of the given element

5.1.4.6 ds_sll_node_t* ds_sll_createNode (void * *element*)

Create a Node to be used with a singly linked list.

Parameters

element	The element to store in the new node

Returns

A singly linked list compatible node containing the given element, or NULL if error occurred

5.1.4.7 void ds_sll_deleteElement (void * *element*) [inline]

Delete an element contained within a node and free any allocated resources.

Parameters

element	The element

5.1.4.8 void ds_sll_deleteNode (ds_sll_node_t * *node*) [inline]

Delete the given node; Free all allocated resources.

Parameters

node	The node to delete

5.1.4.9 ds_sll_error_t ds_sll_deleteNodeAtIndex (ds_sll_t * *linkedList*, int *index*)

Delete the node at the given index from the singly linked list.

Parameters

linkedList	The singly linked list to delete the node from
index	The index of the node to delete

Returns

ds_sll_error_t Error code representing the status of the function

5.1.4.10 ds_sll_error_t ds_sll_destroySinglyLinkedList (ds_sll_t * *linkedList*)

Destroy a Singly Linked List.

Parameters

linkedList	The singly linked list to destroy

Returns

ds_sll_error_t Error code representing the status of the function This function will delete all the nodes and deallocate all related memory. Warning, do not use this function if you are sharing any nodes with another list that is currently in use

5.1.4.11 int ds_sll_executeFunctionOnElements (ds_sll_t * *linkedList*, ds_sll_func_return_t(*)(void *, ds_sll_node_t *, int, void *) *func*, void * *sharedData*)

Executes a function on each element in the linked list in order.

Parameters

linkedList	The singly linked list to map the function to
func	A function to execute on each element. Should be in the form: int func(void* element) It should take one void pointer argument, and return a 1 if an error occurred or a 0 otherwise.
sharedData	A pointer that is passed to your function that you can use to share data to and from your function, such as having one of the function calls return a value via this pointer, or passing an extra parameter to your function.

Returns

-1 if no error occurred; the index of the node where the error occurred at otherwise.

This function traverses the linked list and for each node it traverses, executes the given function passing the element contained in the current node as a parameter. If the function returns DS_SLL_FUNCTION_EXECUTION_ERROR,

or a null node is reached before the entire list has been traversed (reached the tail), then this function will halt and return the index of the node where the error was generated. Otherwise if the function successfully traversed the entire list, or one of the function calls returned DS_SLL_STOP_EXECUTION then the function will halt and return -1 indicating successful execution.

Your given function will be called on each node in sequence (starting from the head) until the tail or until one of the function calls returns DS_SLL_STOP_EXECUTION or DS_SLL_FUNCTION_EXECUTION_ERROR

5.1.4.12 void* ds_sll_extractElementFromNode (ds_sll_node_t * *node*) [inline]

Extract the element contained in the given node.

Parameters

node	The node that contains the element you wish to extract

Returns

The element contained within the given node (void *)

5.1.4.13 ds_sll_node_t* ds_sll_findNodeContainingElement (ds_sll_t * *linkedList,* void * *element,* int(*)(void *, void *) *equalityFunc,* int * *resultIndex*)

Find the node(s) containing the given element.

Parameters

linkedList	The linkedList to search in.
element	The element to search for, If set to NULL, the function will continue the search from the where the previous call left off, (used if the list contains multiple instances of the same element and you wish to iterate through them).
equalityFunc	A function that compares two elements and returns 1 if equal and 0 if not equal.
resultIndex	Optional parameter, if not NULL will be set to equal the index of the node that was found to contain the given element.

Returns

The node containing the given element. NULL if not found or error occurred.

This function can be used either to retrieve one or more nodes containing the given element, (by calling the function once by passing the element that you are searching for, and setting the element parameter to NULL to continue searching for other nodes after the previously found node), and it can also be used to get the index of the node containing the desired element via the resultIndex parameter.

5.1.4.14 void* ds_sll_getElementAtIndex (const ds_sll_t * *linkedList,* int *index*)

Get the element contained in the node at the given index in the specified singly linked list.

Parameters

linkedList	The singly linked list to get the node from
index	The index of the node you want to get (starting with 0)

Returns

A pointer to the element contained by the node at the given index, or NULL if not found or an error occurred.

5.1.4.15 ds_sll_node_t* ds_sll_getNodeAtIndex (const ds_sll_t * *linkedList*, int *index*)

Get the node from a singly linked list at a given index.

Parameters

linkedList	The singly linked list to get the node from
index	The index of the node you want to get (starting with 0)

Returns

A pointer to the node at the given index, or NULL if not found or an error occurred.

5.1.4.16 ds_sll_error_t ds_sll_insertElementAtIndex (ds_sll_t * *linkedList*, void * *element*, int *index*)

Create a new node with the given element and inserts the new node at the chosen index.

Parameters

linkedList	The singly linked list to insert the node into
element	The element pointer you wish to store in the node
index	The index where the node should be inserted

Returns

An error code indicating the completion status of the function

See also

ds_sll_error_t

5.1.4.17 ds_sll_error_t ds_sll_insertElementCopyAtIndex (ds_sll_t * *linkedList*, void * *element*, const size_t *element_size*, int *index*)

Create a new node with a copy of the given element and inserts the new node at the chosen index.

Parameters

linkedList	The singly linked list to insert the node into
element	A pointer to the element that you wish to store a copy of in the new node
element_size	The size (in bytes) of the given element
index	The index where the node should be inserted

Returns

An error code indicating the completion status of the function

See also

ds_sll_error_t

5.1.4.18 ds_sll_error_t ds_sll_insertNodeAtIndex (ds_sll_t * *linkedList*, ds_sll_node_t * *node*, int *index*)

Inserts the given node at the chosen index.

Parameters

linkedList	The singly linked list to insert the node into
node	The node to insert
index	The index where the node should be inserted

Returns

An error code indicating the completion status of the function

See also

ds_sll_error_t

5.1.4.19 ds_sll_t* ds_sll_newSinglyLinkedList ( )

Create a new singly linked list.

Returns

Returns a pointer to a new Singly Linked List struct (linked list header)

5.1.4.20 ds_sll_node_t* ds_sll_nextNode (ds_sll_node_t * *node*) [inline]

Inline function that returns the next node in the given list.

Parameters

node	The node to extract the 'next node' from

Returns

The next node located right after the given node

The point of this function is to abstract the node->next operation allowing the developer to easily customize the way linked lists are traversed

5.1.4.21 ds_sll_error_t ds_sll_splitSinglyLinkedListAtIndex (ds_sll_t * *firstLinkedList,* ds_sll_t * *secondLinkedList,* int *index*)

Split Singly Linked List into two singly linked lists at the given index.

Parameters

firstLinkedList	The original Singly Linked List to be split, will be updated to point to the first sublist created (must have len > 1).
secondLinkedList	A new (initialized) Singly Linked List header to be set to point to the second sublist created
index	The index where the singly linked list will be split (the new tail of the current singly linked list).

Returns

ds_sll_error_t Error Code.

The function will shorten the given singly linked list so that it's tail now points to the node located at the given index, and it will create a new singly linked list header where it's head will point to the node at right after the given index (node at index + 1) and it's tail will point to the tail of the original given linked list. After the function completes successfully: The firstLinkedList will point to the sublist starting with the original head up to the node specified by the given index. The secondLinkedList will point to the sublist starting with the node right after the node at the given index up to the original tail. It is expected that you pass a new singly linked list header as the secondLinkedList parameter

5.1.4.22 void ds_sll_storeElementInNode (ds_sll_node_t * *node,* void * *element*) [inline]

Encapsulate an element within a given node.

Parameters

node	The node to store the element in
element	The element to store in the node The reason that I have abstracted this function is to allow the programmer to change the way data is being stored and extracted within nodes. Note that at the moment of this writing, the nodes store void pointers. Though you may use those pointers as a gate to more complex operations as you see fit.

5.1.4.23 ds_sll_error_t ds_sll_traverseNodeToIndex (const ds_sll_t * *linkedList*, ds_sll_node_t ** *node*, int *index*)

Taverse a linked list to the node at a given index.

Parameters

linkedList	The singly linked list to traverse
node	A free node pointer to traverse with
index	The index to traverse to

Returns

ds_sll_error_t Error Code.

This function will traverse the given list using the given node pointer until the given index. The functions also returns an error code indicating the success status. This functions intended purpose is to be used with other functions in this library that require traversing a singly linked list. Basically it has the side effect of setting the given node pointer to point to the desired node (if no error occurred) and also returns an error code indicating the status of the traversal.

Chapter 6

Data Structure Documentation

6.1 ds_sll_node_t Struct Reference

`#include <SinglyLinkedList.h>`

Collaboration diagram for ds_sll_node_t:

Data Fields

- void * element
- struct ds_sll_node_t * next

6.1.1 Detailed Description

Singly Linked List Node datatype

6.1.2 Field Documentation

6.1.2.1 void* ds_sll_node_t::element

Pointer to the data being stored in the node. User is responsible for typecasting this pointer appropriately

6.1.2.2 struct ds_sll_node_t* ds_sll_node_t::next

pointer to the next node in the list

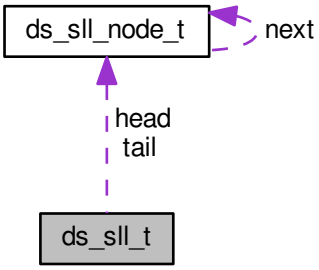
The documentation for this struct was generated from the following file:

- src/SinglyLinkedList.h

6.2 ds_sll_t Struct Reference

`#include <SinglyLinkedList.h>`

Collaboration diagram for ds_sll_t:



Data Fields

- ds_sll_node_t * head
- ds_sll_node_t * tail

6.2.1 Detailed Description

Singly Linked List datatype. This struct holds the information that identifies a Singly Linked List

6.2.2 Field Documentation

6.2.2.1 ds_sll_node_t* ds_sll_t::head

pointer to the first node in the linked list

6.2.2.2 ds_sll_node_t* ds_sll_t::tail

pointer to the last node in the linked list

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

- src/SinglyLinkedList.h

Chapter 7

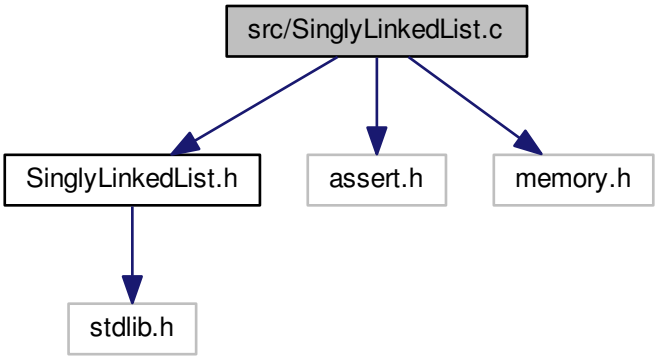
File Documentation

7.1 src/SinglyLinkedList.c File Reference

Singly Linked List library (ds_sll)

```
#include "SinglyLinkedList.h"
#include <assert.h>
#include <memory.h>
```

Include dependency graph for SinglyLinkedList.c:



Macros

- #define ASSERT assert

  Macro definition for ASSERT Used to enforce Design by Contract coding Typically disabled on release.

Functions

- ds_sll_t * ds_sll_newSinglyLinkedList ()

  Create a new singly linked list.
- ds_sll_node_t * ds_sll_createNode (void *element)

  Create a Node to be used with a singly linked list.
- void * ds_sll_extractElementFromNode (ds_sll_node_t *node)

  Extract the element contained in the given node.
- void ds_sll_storeElementInNode (ds_sll_node_t *node, void *element)

  Encapsulate an element within a given node.
- ds_sll_node_t * ds_sll_nextNode (ds_sll_node_t *node)

  Inline function that returns the next node in the given list.
- void ds_sll_deleteElement (void *element)

  Delete an element contained within a node and free any allocated resources.
- void ds_sll_deleteNode (ds_sll_node_t *node)

  Delete the given node; Free all allocated resources.
- void * ds_sll_copyElement (void *element, const size_t element_size)

  Copy a given element to be stored in a node.
- ds_sll_node_t * ds_sll_getNodeAtIndex (const ds_sll_t *linkedList, int index)

  Get the node from a singly linked list at a given index.
- ds_sll_error_t ds_sll_traverseNodeToIndex (const ds_sll_t *linkedList, ds_sll_node_t **node, int index)

  Taverse a linked list to the node at a given index.
- void * ds_sll_getElementAtIndex (const ds_sll_t *linkedList, int index)

  Get the element contained in the node at the given index in the specified singly linked list.
- ds_sll_error_t ds_sll_deleteNodeAtIndex (ds_sll_t *linkedList, int index)

  Delete the node at the given index from the singly linked list.
- ds_sll_error_t ds_sll_destroySinglyLinkedList (ds_sll_t *linkedList)

  Destroy a Singly Linked List.
- void ds_sll_appendNode (ds_sll_t *linkedList, ds_sll_node_t *node)

  Append a given node to the end of the given singly linked list.
- ds_sll_error_t ds_sll_appendElement (ds_sll_t *linkedList, void *element)

  Create a new node encapsulating the given element, and append this node to the end of the given singly linked list.
- ds_sll_error_t ds_sll_appendElementCopy (ds_sll_t *linkedList, void *element, const size_t element_size)

  Create a new node encapsulating a copy of the given element, and append this node to the end of the given singly linked list.
- ds_sll_error_t ds_sll_insertNodeAtIndex (ds_sll_t *linkedList, ds_sll_node_t *node, int index)

  Inserts the given node at the chosen index.
- ds_sll_error_t ds_sll_insertElementAtIndex (ds_sll_t *linkedList, void *element, int index)

  Create a new node with the given element and inserts the new node at the chosen index.
- ds_sll_error_t ds_sll_insertElementCopyAtIndex (ds_sll_t *linkedList, void *element, const size_t element_size, int index)

  Create a new node with a copy of the given element and inserts the new node at the chosen index.
- int ds_sll_executeFunctionOnElements (ds_sll_t *linkedList, ds_sll_func_return_t(*func)(void *, ds_sll_node_t *, int, void *), void *sharedData)

  Executes a function on each element in the linked list in order.
- int ds_sll_calculateLength (const ds_sll_t *linkedList)

  Calculates the length of the singly linked list (by traversing it)
- ds_sll_error_t ds_sll_splitSinglyLinkedListAtIndex (ds_sll_t *firstLinkedList, ds_sll_t *secondLinkedList, int index)

  Split Singly Linked List into two singly linked lists at the given index.
- ds_sll_node_t * ds_sll_findNodeContainingElement (ds_sll_t *linkedList, void *element, int(*equalityFunc)(void *, void *), int *resultIndex)

  Find the node(s) containing the given element.

7.1.1 Detailed Description

Singly Linked List library (ds_sll)

Category: Data Structures >> Linked Lists Codename: ds_sll

Please familiarize yourself with the fundamental datatypes used throughout: ds_sll_t ds_sll_node_t

Usage:

This version of singly linked list does not support multiple linked lists being combined manually. eg: appending a node that already exists in the middle of list A to the tail of list B Doing such an operation by logic should automatically mean that list A now contains the same nodes as list B after the common node is traversed, but this version of the data structure can't know where the new tail resides if you do the aforementioned operation as it keeps track of both the head and the tail, and hence you will need to manually update the tail to point to the correct node, which in this case is the tail of list A. An alternative is to use the combine function which deals with the nuisance of correcting the tail pointers appropriately. Also there are plans to implement a headless linked list (a linked list that does not keep track of the head and tail but simply uses any given node to traverse) in the future which allows much more flexibility but at the expense of not knowing whether or not the list is broken or corrupt.

Note:

I follow DbC (Design by Contract) with my code, meaning functions that are called with illogical or invalid arguments will cause the program to abrupt using the assert functionality. Feel free to redefine how assert works to suite your projects needs, or even nullify all the asserts with a macro definition to NDEBUG #define NDEBUG

See also

ds_sll_t
ds_sll_node_t

Operations:

- New: create a new Singly Linked List
- Destroy: delete all elements from the Singly Linked List and free all resources
- Append Element: Create a new node with the given element and append it to the linked list
- Create Node: Create a new node with the given element
- Insert Node: Insert a given node at the specified index
- Append Node: Append a given node to the end of the linked list
- Delete Node: Delete the node at the given index
- Get Node: Get the node at the given index
- Extract Element From Node: Extract the element from a given node
- Find: Find an element in the linked list using the given equality function
- Split: Split the linked list at the given index
- Execute Function on Elements: Executes the given function on the element of every node
- Length Of: Get the length of the linked list

Abstractions:

This library was designed with some abstractions in mind to allow for easy customization according to your project's needs. These abstractions are mainly implemented as inline functions where possible. The main abstractions are:

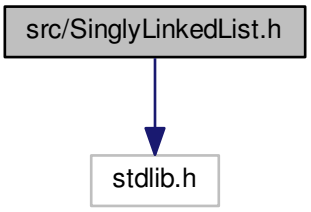
- createNode: allows you to specify how nodes are created
- extractElementFromNode: allows you to specify how node elements are read (return node->element)
- storeElementInNode: allows you to specify how to insert an element in a node (node->element = new_↩ element)
- deleteElement: allows you to specify how to deallocate the resources of a given Element
- deleteNode: allows you to specify how a node is 'deleted' and its resources are de-allocated
- copyElement: allows you to specify how an Element is copied You can alter these functions along with the node data type to suite the needs of your application

7.2 src/SinglyLinkedList.h File Reference

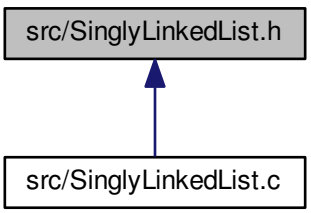
Singly Linked List library (Header) (ds_sll)

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

Include dependency graph for SinglyLinkedList.h:



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



Data Structures

- struct ds_sll_node_t
- struct ds_sll_t

Typedefs

- typedef struct ds_sll_node_t ds_sll_node_t
- typedef struct ds_sll_t ds_sll_t
- typedef enum ds_sll_error_t ds_sll_error_t
- typedef enum ds_sll_func_return_t ds_sll_func_return_t

Enumerations

- enum ds_sll_error_t { DS_SLL_NO_ERROR = 0, DS_SLL_NODE_CREATION_ERROR, DS_SLL_ELEMENT_CREATION_ERROR, DS_SLL_LIST_CREATION_ERROR, DS_SLL_INDEX_OUT_OF_BOUNDS_ERROR, DS_SLL_BROKEN_LIST_ERROR, DS_SLL_LIST_TOO_SMALL_ERROR, DS_SLL_FUNCTION_EXECUTION_ERROR }
- enum ds_sll_func_return_t { DS_SLL_CONTINUE_EXECUTION, DS_SLL_STOP_EXECUTION, DS_SLL_EXECUTION_ERROR }

Functions

- ds_sll_t * ds_sll_newSinglyLinkedList ()

  Create a new singly linked list.
- ds_sll_node_t * ds_sll_createNode (void *element)

  Create a Node to be used with a singly linked list.
- ds_sll_error_t ds_sll_destroySinglyLinkedList (ds_sll_t *linkedList)

  Destroy a Singly Linked List.
- void ds_sll_deleteNode (ds_sll_node_t *node)

  Delete the given node; Free all allocated resources.
- ds_sll_error_t ds_sll_deleteNodeAtIndex (ds_sll_t *linkedList, int index)

  Delete the node at the given index from the singly linked list.
- ds_sll_node_t * ds_sll_nextNode (ds_sll_node_t *node)

  Inline function that returns the next node in the given list.
- void * ds_sll_extractElementFromNode (ds_sll_node_t *node)

  Extract the element contained in the given node.
- void ds_sll_storeElementInNode (ds_sll_node_t *node, void *element)

  Encapsulate an element within a given node.
- void ds_sll_deleteElement (void *node)

  Delete an element contained within a node and free any allocated resources.
- void * ds_sll_copyElement (void *element, const size_t element_size)

  Copy a given element to be stored in a node.
- int ds_sll_executeFunctionOnElements (ds_sll_t *linkedList, ds_sll_func_return_t(*func)(void *, ds_sll_node_t *, int, void *), void *sharedData)

  Executes a function on each element in the linked list in order.
- int ds_sll_calculateLength (const ds_sll_t *linkedList)

  Calculates the length of the singly linked list (by traversing it)

- ds_sll_error_t ds_sll_splitSinglyLinkedListAtIndex (ds_sll_t *firstLinkedList, ds_sll_t *secondLinkedList, int index)

  Split Singly Linked List into two singly linked lists at the given index.
- ds_sll_node_t * ds_sll_getNodeAtIndex (const ds_sll_t *linkedList, int index)

  Get the node from a singly linked list at a given index.
- void * ds_sll_getElementAtIndex (const ds_sll_t *linkedList, int index)

  Get the element contained in the node at the given index in the specified singly linked list.
- ds_sll_node_t * ds_sll_findNodeContainingElement (ds_sll_t *linkedList, void *element, int(*equality↩ Func)(void *, void *), int *resultIndex)

  Find the node(s) containing the given element.
- void ds_sll_appendNode (ds_sll_t *linkedList, ds_sll_node_t *node)

  Append a given node to the end of the given singly linked list.
- ds_sll_error_t ds_sll_appendElement (ds_sll_t *linkedList, void *element)

  Create a new node encapsulating the given element, and append this node to the end of the given singly linked list.
- ds_sll_error_t ds_sll_appendElementCopy (ds_sll_t *linkedList, void *element, const size_t element_size)

  Create a new node encapsulating a copy of the given element, and append this node to the end of the given singly linked list.
- ds_sll_error_t ds_sll_insertNodeAtIndex (ds_sll_t *linkedList, ds_sll_node_t *node, int index)

  Inserts the given node at the chosen index.
- ds_sll_error_t ds_sll_insertElementAtIndex (ds_sll_t *linkedList, void *element, int index)

  Create a new node with the given element and inserts the new node at the chosen index.
- ds_sll_error_t ds_sll_insertElementCopyAtIndex (ds_sll_t *linkedList, void *element, const size_t element↩ _size, int index)

  Create a new node with a copy of the given element and inserts the new node at the chosen index.
- ds_sll_error_t ds_sll_traverseNodeToIndex (const ds_sll_t *linkedList, ds_sll_node_t **node, int index)

  Taverse a linked list to the node at a given index.

7.2.1 Detailed Description

Singly Linked List library (Header) (ds_sll)

Category: Data Structures >> Linked Lists >> Singly Linked List Codename: ds_sll

The main two data types that the user should be aware off: ds_sll_t ds_sll_node_t

Index