cdatastruct

Generated by Doxygen 1.8.1.2

Fri Sep 6 2013 20:12:51

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

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

Chapter 3

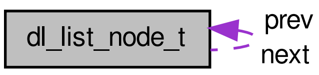
Data Structure Documentation

3.1 dl_list_node_t Struct Reference

Struct for double linked list node.

`#include <cds_dl_list.h>`

Collaboration diagram for dl_list_node_t:



Data Fields

- void ∗ data
- struct dl_list_node_t ∗ next
- struct dl_list_node_t ∗ prev

3.1.1 Field Documentation

3.1.1.1 void∗ dl_list_node_t::data

Pointer to data

3.1.1.2 struct dl_list_node_t∗ dl_list_node_t::next

Pointer to next node

3.1.1.3 struct dl_list_node_t∗ dl_list_node_t::prev

Pointer to previous node

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

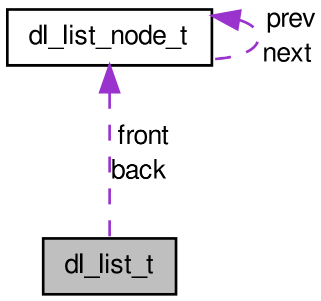
- cds_dl_list.h

3.2 dl_list_t Struct Reference

Struct to contain a list.

```
#include <dl_list.h>
```

Collaboration diagram for dl_list_t:



Data Fields

- struct dl_list_node_t ∗ front
- struct dl_list_node_t ∗ back
- size_t length
- int(∗ cfunc)()

3.2.1 Field Documentation

3.2.1.1 struct dl_list_node_t∗ dl_list_t::back

Pointer to last node

3.2.1.2 int(∗ dl_list_t::cfunc)()

Pointer to compare function

3.2.1.3 struct dl_list_node_t∗ dl_list_t::front

Pointer to first node

3.2.1.4 size_t dl_list_t::length

Length of list

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

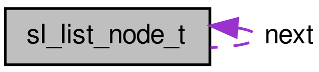
- dl_list.h

3.3 sl_list_node_t Struct Reference

Struct for singly linked list node.

`#include <cds_sl_list.h>`

Collaboration diagram for sl_list_node_t:



Data Fields

- void ∗ data
- struct sl_list_node_t ∗ next

3.3.1 Field Documentation

3.3.1.1 void∗ sl_list_node_t::data

Pointer to data

3.3.1.2 struct sl_list_node_t∗ sl_list_node_t::next

Pointer to next node

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

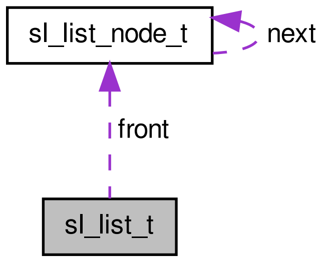
- cds_sl_list.h

3.4 sl_list_t Struct Reference

Struct to contain a list.

`#include <sl_list.h>`

Collaboration diagram for sl_list_t:



Data Fields

- struct sl_list_node_t ∗ front
- size_t length
- int(∗ cfunc)()

3.4.1 Field Documentation

3.4.1.1 int(∗ sl_list_t::cfunc)()

Pointer to compare function

3.4.1.2 struct sl_list_node_t∗ sl_list_t::front

Pointer to first node

3.4.1.3 size_t sl_list_t::length

Length of list

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

- sl_list.h

Chapter 4

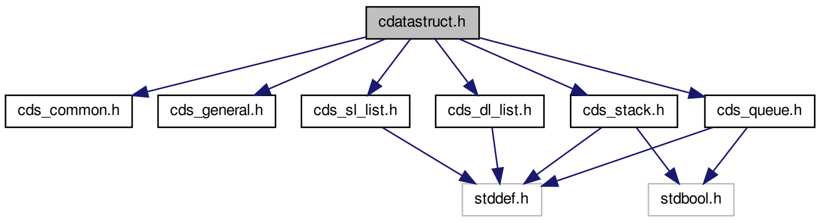
File Documentation

4.1 cdatastruct.h File Reference

Interface to generic C data structures.

```
#include "cds_common.h"
#include "cds_general.h"
#include "cds_sl_list.h"
#include "cds_dl_list.h"
#include "cds_stack.h"
#include "cds_queue.h"
```

Include dependency graph for cdatastruct.h:



4.1.1 Detailed Description

Interface to generic C data structures.

Author

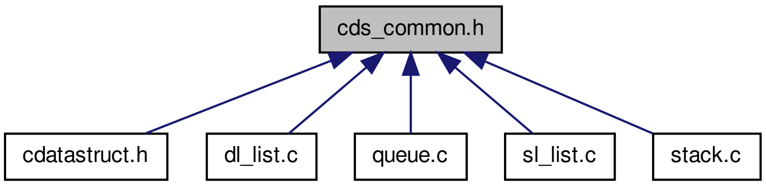
Paul Griffiths

Copyright

4.2 cds_common.h File Reference

Common data types and data for C data structures library.

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



Typedefs

- typedef enum cds_error cds_error

    Enumeration of return error codes.

Enumerations

- enum cds_error { CDSERR_ERROR = -1, CDSERR_OUTOFRANGE = -2, CDSERR_NOTFOUND = -3, C-DSERR_BADITERATOR = -4 }

    Enumeration of return error codes.

4.2.1 Detailed Description

Author

Paul Griffiths

Copyright

Copyright 2013 Paul Griffiths. Distributed under the terms of the GNU General Public License. http-://www.gnu.org/licenses/

4.2.2 Enumeration Type Documentation

4.2.2.1 enum **cds_error**

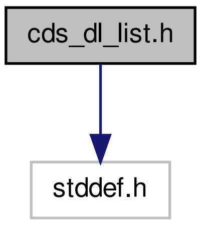
Enumerator:

CDSERR_ERROR	Unspecified error
CDSERR_OUTOFRANGE	Index out of range
CDSERR_NOTFOUND	Data element not found
CDSERR_BADITERATOR	Invalid iterator

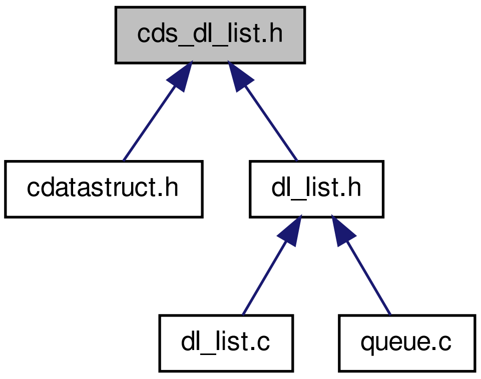
4.3 cds_dl_list.h File Reference

User interface to doubly linked list data structure.

`#include <stddef.h>`
Include dependency graph for cds_dl_list.h:



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



Data Structures

- struct dl_list_node_t

  Struct for double linked list node.

Typedefs

- typedef struct dl_list_node_t dl_list_node_t

  Struct for double linked list node.
- typedef struct dl_list_t ∗ dl_list

  Typedef for list pointer.
- typedef struct dl_list_node_t ∗ dl_list_itr

  Typedef for list iterator.

Functions

- dl_list dl_list_init (int(∗cfunc)(const void ∗, const void ∗))

    Initializes a new doubly linked list.
- void dl_list_free (dl_list list)

    Frees the resources associated with a list.
- size_t dl_list_length (const dl_list list)

    Returns the number of elements in a list.
- bool dl_list_isempty (const dl_list list)

    Checks if a list is empty.
- void dl_list_prepend (dl_list list, void ∗data)

    Inserts an element at the beginning of a list.
- void dl_list_append (dl_list list, void ∗data)

    Inserts an element at the end of a list.
- int dl_list_insert_before (dl_list list, const dl_list_itr itr, void ∗data)

    Inserts an element before a provided iterator.
- int dl_list_insert_at (dl_list list, const size_t index, void ∗data)

    Inserts an element at the specified index of a list.
- int dl_list_insert_after (dl_list list, const dl_list_itr itr, void ∗data)

    Inserts an element after a provided iterator.
- int dl_list_delete_at (dl_list list, const size_t index)

    Deletes a list element at a specified index.
- int dl_list_find_index (const dl_list list, const void ∗data)

    Finds the index of the specified data in a list.
- dl_list_itr dl_list_find_itr (const dl_list list, const void ∗data)

    Gets an iterator to the specified data in a list.
- void ∗ dl_list_data (const dl_list list, const size_t index)

    Returns a pointer to the data at a specified index.
- dl_list_itr dl_list_first (const dl_list list)

    Returns an iterator to the first element of a list.
- dl_list_itr dl_list_last (const dl_list list)

    Returns an iterator to the last element of a list.
- dl_list_itr dl_list_next (const dl_list_itr itr)

    Advances a list iterator by one element.
- dl_list_itr dl_list_prev (const dl_list_itr itr)

    Backs up a list iterator by one element.
- dl_list_itr dl_list_itr_from_index (const dl_list list, const size_t index)

    Return an iterator to a specified element of a list.

4.3.1 Detailed Description

Author

Paul Griffiths

Copyright

Copyright 2013 Paul Griffiths. Distributed under the terms of the GNU General Public License. `http-://www.gnu.org/licenses/`

4.3.2 Function Documentation

4.3.2.1 void dl_list_append (dl_list *list*, void * *data*)

Parameters

list	A pointer to the list.
data	A pointer to the data to add. The memory pointed to by this parameter must be dynamically allocated, as an attempt will be made to `free()` it when deleting the list.

4.3.2.2 void* dl_list_data (const dl_list *list*, const size_t *index*)

Parameters

list	A pointer to the list.
index	The index of the data.

Returns

A pointer to the data, or NULL if the index is out of range.

4.3.2.3 int dl_list_delete_at (dl_list *list*, const size_t *index*)

Parameters

list	A pointer to the list.
index	The index of the element to delete.

Returns

0 on success, CDSERR_OUTOFRANGE if the the index is out of range.

4.3.2.4 int dl_list_find_index (const dl_list *list*, const void * *data*)

Parameters

list	A pointer to the list.
data	A pointer to the data to find.

Returns

The index of the element, if found, or CDSERR_NOTFOUND if it is not in the list.

4.3.2.5 dl_list_itr dl_list_find_itr (const dl_list *list*, const void * *data*)

Parameters

list	A pointer to the list.
data	A pointer to the data to find.

Returns

An iterator to the found element, or NULL is the element is not in the list.

4.3.2.6 dl_list_itr dl_list_first (const dl_list *list*)

Parameters

list	A pointer to the list.

Returns

An iterator to the first element.

4.3.2.7 void dl_list_free (dl_list *list*)

Parameters

list	A pointer to the list to free.

4.3.2.8 dl_list dl_list_init (int(*)(const void *, const void *) *cfunc*)

Parameters

cfunc	A pointer to a compare function. The function should return `int` and accept two parameters of type `void *`. It should return less than 1 if the first parameter is less than the second, greater than 1 if the first parameter is greater than the second, and zero if the parameters are equal.

Returns

A pointer to the new list.

4.3.2.9 int dl_list_insert_after (dl_list *list,* const dl_list_itr *itr,* void * *data*)

Note that dl_list_first() may return a NULL iterator when the list is empty. One reasonable behavior for this function would be to add a new node to the list in that case. However, an iterator may also become NULL when advanced to the end of the list. One possible way to modify this function would be to check the length of this list when the iterator is NULL, and if it is zero, add the first node to the list. However, the semantic meaning of adding an element *after* an iterator breaks down if that that iterator does not point to an existing element. Therefore, it is simpler for this function to simply refuse to handle NULL iterators. It is unlikely a user would want to call this function unless there are already elements in a list, and a valid iterator has been returned, e.g. through a find function.

Parameters

list	A pointer to the list.
itr	The iterator after which to insert.
data	A pointer to the data to add. The memory pointed to by this parameter must be dynamically allocated, as an attempt will be made to `free()` it when deleting the list.

Returns

0 on success, CDSERR_BADITERATOR if `itr` is a NULL pointer.

4.3.2.10 int dl_list_insert_at (dl_list *list,* const size_t *index,* void * *data*)

Parameters

list	A pointer to the list.
index	The index at which to insert. Setting this equal to the length of the list (i.e. to one element past the zero-based index of the last element) inserts the element at the end of the list.
data	A pointer to the data to add. The memory pointed to by this parameter must be dynamically allocated, as an attempt will be made to `free()` it when deleting the list.

Returns

0 on success, CDSERR_OUTOFRANGE if `index` exceeds the length of the list.

4.3.2.11 int dl_list_insert_before (**dl_list** *list,* const **dl_list_itr** *itr,* void ∗ *data*)

Parameters

list	A pointer to the list.
itr	The iterator after which to insert.
data	A pointer to the data to add. The memory pointed to by this parameter must be dynamically allocated, as an attempt will be made to `free()` it when deleting the list.

Returns

0 on success, CDSERR_BADITERATOR if `itr` is a NULL pointer.

4.3.2.12 bool dl_list_isempty (const **dl_list** *list*)

Parameters

list	A pointer to the list.

Returns

`true` if the list is empty, otherwise `false`.

4.3.2.13 **dl_list_itr** dl_list_itr_from_index (const **dl_list** *list,* const size_t *index*)

Parameters

list	A pointer to the list.
index	The specified index.

Returns

The iterator, or NULL if `index` is out of range.

4.3.2.14 **dl_list_itr** dl_list_last (const **dl_list** *list*)

Parameters

list	A pointer to the list.

Returns

An iterator to the first element.

4.3.2.15 size_t dl_list_length (const **dl_list** *list*)

Parameters

list	A pointer to the list.

4.3.2.16 **dl_list_itr** dl_list_next (const **dl_list_itr** *itr*)

Parameters

itr	The iterator to advance

Returns

The advanced iterator.

4.3.2.17 void dl_list_prepend (**dl_list** *list,* void * *data*)

Parameters

list	A pointer to the list.
data	A pointer to the data to add. The memory pointed to by this parameter must be dynamically allocated, as an attempt will be made to `free()` it when deleting the list.

4.3.2.18 **dl_list_itr** dl_list_prev (const **dl_list_itr** *itr*)

Parameters

itr	The iterator to back up.

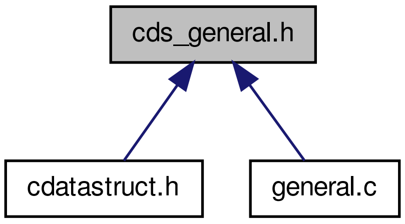
Returns

The backed up iterator.

4.4 cds_general.h File Reference

Interface to general data structure helper functions.

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



Functions

- void ∗ cds_new_int (const int n)

  Dynamically allocates memory for a new `int`*.*
- void ∗ cds_new_uint (const unsigned int n)

  Dynamically allocates memory for a new `unsigned int`*.*
- void ∗ cds_new_long (const long n)

  Dynamically allocates memory for a new `long`*.*
- void ∗ cds_new_ulong (const unsigned long n)

  Dynamically allocates memory for a new `unsigned long`*.*
- void ∗ cds_new_string (const char ∗str)

  Dynamically allocates memory for a new string.
- int cds_compare_int (const void ∗data, const void ∗cmp)

  Compares two `int` *via* `void` *pointers.*
- int cds_compare_uint (const void ∗data, const void ∗cmp)

  Compares two `unsigned int` *via* `void` *pointers.*
- int cds_compare_long (const void ∗data, const void ∗cmp)

  Compares two `long` *via* `void` *pointers.*
- int cds_compare_ulong (const void ∗data, const void ∗cmp)

  Compares two `unsigned long` *via* `void` *pointers.*
- int cds_compare_string (const void ∗data, const void ∗cmp)

  Compares two strings via `void` *pointers.*

4.4.1 Detailed Description

Interface to general data structure helper functions.

Author

Paul Griffiths

Copyright

4.4.2 Function Documentation

4.4.2.1 int cds_compare_int (const void ∗ *data*, const void ∗ *cmp*)

Parameters

data	Pointer to the data to which to compare.
cmp	Pointer to the comparison data.

Returns

-1 if the comparison data is greater than the data, 1 if the comparison data is less than the data, and 0 if the comparison data is equal to the data.

4.4.2.2 int cds_compare_long (const void ∗ *data*, const void ∗ *cmp*)

Parameters

data	Pointer to the data to which to compare.
cmp	Pointer to the comparison data.

Returns

-1 if the comparison data is greater than the data, 1 if the comparison data is less than the data, and 0 if the comparison data is equal to the data.

4.4.2.3 int cds_compare_string (const void ∗ *data*, const void ∗ *cmp*)

Parameters

data	Pointer to the data to which to compare.
cmp	Pointer to the comparison data.

Returns

-1 if the comparison data is greater than the data, 1 if the comparison data is less than the data, and 0 if the comparison data is equal to the data.

4.4.2.4 int cds_compare_uint (const void ∗ *data*, const void ∗ *cmp*)

Parameters

data	Pointer to the data to which to compare.
cmp	Pointer to the comparison data.

Returns

-1 if the comparison data is greater than the data, 1 if the comparison data is less than the data, and 0 if the comparison data is equal to the data.

4.4.2.5 int cds_compare_ulong (const void ∗ *data*, const void ∗ *cmp*)

Parameters

data	Pointer to the data to which to compare.
cmp	Pointer to the comparison data.

Returns

-1 if the comparison data is greater than the data, 1 if the comparison data is less than the data, and 0 if the comparison data is equal to the data.

4.4.2.6 void∗ cds_new_int (const int *n*)

Parameters

n	The new `int` for which to allocate.

Returns

A `void` pointer to the allocated memory.

4.4.2.7 void∗ cds_new_long (const long *n*)

Parameters

n	The new `long` for which to allocate.

Returns

A `void` pointer to the allocated memory.

4.4.2.8 void∗ cds_new_string (const char ∗ *str*)

Parameters

str	The new string for which to allocate.

Returns

A `void` pointer to the allocated memory.

4.4.2.9 void∗ cds_new_uint (const unsigned int *n*)

Parameters

n	The new `unsigned int` for which to allocate.

Returns

A `void` pointer to the allocated memory.

4.4.2.10 void∗ cds_new_ulong (const unsigned long *n*)

Parameters

n	The new `unsigned long` for which to allocate.

Returns

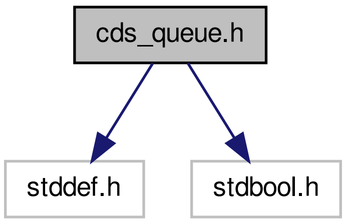
A `void` pointer to the allocated memory.

4.5 cds_queue.h File Reference

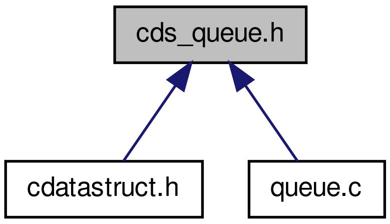
User interface to queue data structure.

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

Include dependency graph for cds_queue.h:



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



Typedefs

- typedef struct dl_list_t ∗ queue

  Typedef for queue pointer.

Functions

- queue queue_init (void)

    Initializes a new queue.

- void queue_free (queue que)

    Frees memory and releases resources used by a queue.

- size_t queue_length (const queue que)

    Gets the number of items in a queue.

- bool queue_isempty (const queue que)

    Checks if a queue is empty.

- void ∗ queue_pop (queue que)

    Pops a data item from the queue.

- void queue_pushback (queue que, void ∗data)

    Pushes a data item onto the back of the queue.

4.5.1 Detailed Description

Author

Paul Griffiths

Copyright

Copyright 2013 Paul Griffiths. Distributed under the terms of the GNU General Public License. `http://www.gnu.org/licenses/`

4.5.2 Function Documentation

4.5.2.1 void queue_free (queue *que*)

Parameters

que	A pointer to the queue.

4.5.2.2 queue queue_init (void)

Returns

A pointer to the new queue.

4.5.2.3 bool queue_isempty (const queue *que*)

Parameters

que	A pointer to the queue.

Returns

`true` is the queue is empty, `false` if not.

4.5.2.4 size_t queue_length (const queue *que*)

Parameters

que	A pointer to the queue.

Returns

The number of items in the queue.

4.5.2.5 void∗ queue_pop (queue *que*)

The item returned was previously allocated using `malloc()`, so the user must `free()` the returned pointer when done.

Parameters

que	A pointer to the queue.

Returns

A `void` pointer to the popped data item.

4.5.2.6 void queue_pushback (queue *que,* void ∗ *data*)

The provided pointer should point to dynamically allocated memory.

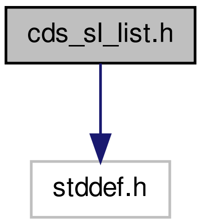
Parameters

que	A pointer to the queue.
data	A pointer to the data item to be pushed.

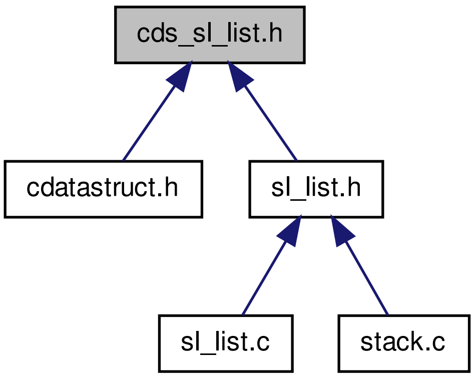
4.6 cds_sl_list.h File Reference

User interface to singly linked list data structure.

`#include <stddef.h>`
Include dependency graph for cds_sl_list.h:



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



Data Structures

- struct sl_list_node_t

  Struct for singly linked list node.

Typedefs

- typedef struct sl_list_node_t sl_list_node_t

  Struct for singly linked list node.
- typedef struct sl_list_t ∗ sl_list

  Typedef for list pointer.
- typedef struct sl_list_node_t ∗ sl_list_itr

  Typedef for list iterator.

Functions

- sl_list sl_list_init (int(∗cfunc)(const void ∗, const void ∗))

  Initializes a new singly linked list.
- void sl_list_free (sl_list list)

  Frees the resources associated with a list.
- size_t sl_list_length (const sl_list list)

  Returns the number of elements in a list.
- bool sl_list_isempty (const sl_list list)

  Checks if a list is empty.
- void sl_list_prepend (sl_list list, void ∗data)

  Inserts an element at the beginning of a list.
- int sl_list_insert_at (sl_list list, const size_t index, void ∗data)

  Inserts an element at the specified index of a list.
- int sl_list_insert_after (sl_list list, const sl_list_itr itr, void ∗data)

  Inserts an element after a provided iterator.

- int sl_list_delete_at (sl_list list, const size_t index)

  Deletes a list element at a specified index.
- int sl_list_find_index (const sl_list list, const void ∗data)

  Gets an index to the specified data in a list.
- sl_list_itr sl_list_find_itr (const sl_list list, const void ∗data)

  Gets an iterator to the specified data in a list.
- void ∗ sl_list_data (const sl_list list, const size_t index)

  Returns a pointer to the data at a specified index.
- sl_list_itr sl_list_first (const sl_list list)

  Returns an iterator to the first element of a list.
- sl_list_itr sl_list_next (const sl_list_itr itr)

  Advances a list iterator by one element.
- sl_list_itr sl_list_itr_from_index (const sl_list list, const size_t index)

  Return an iterator to a specified element of a list.

4.6.1 Detailed Description

Author

Paul Griffiths

Copyright

Copyright 2013 Paul Griffiths. Distributed under the terms of the GNU General Public License. http-://www.gnu.org/licenses/

4.6.2 Function Documentation

4.6.2.1 void∗ sl_list_data (const sl_list *list*, const size_t *index*)

Parameters

list	A pointer to the list.
index	The index of the data.

Returns

A pointer to the data, or NULL if the index is out of range.

4.6.2.2 int sl_list_delete_at (sl_list *list*, const size_t *index*)

Parameters

list	A pointer to the list.
index	The index of the element to delete.

Returns

0 on success, CDSERR_OUTOFRANGE if the the index is out of range.

4.6.2.3 int sl_list_find_index (const sl_list *list*, const void ∗ *data*)

Parameters

list	A pointer to the list.
data	A pointer to the data to find.

Returns

The index of the found element, or CDSERR_NOTFOUND if the element is not in the list.

4.6.2.4 sl_list_itr sl_list_find_itr (const sl_list *list*, const void ∗ *data*)

Parameters

list	A pointer to the list.
data	A pointer to the data to find.

Returns

An iterator to the found element, or NULL is the element is not in the list.

4.6.2.5 sl_list_itr sl_list_first (const sl_list *list*)

Parameters

list	A pointer to the list.

Returns

An iterator to the first element.

4.6.2.6 void sl_list_free (sl_list *list*)

Parameters

list	A pointer to the list to free.

4.6.2.7 sl_list sl_list_init (int(∗)(const void ∗, const void ∗) *cfunc*)

Parameters

cfunc	A pointer to a compare function. The function should return `int` and accept two parameters of type `void *`. It should return less than 1 if the first parameter is less than the second, greater than 1 if the first parameter is greater than the second, and zero if the parameters are equal.

Returns

A pointer to the new list.

4.6.2.8 int sl_list_insert_after (sl_list *list*, const sl_list_itr *itr*, void ∗ *data*)

Parameters

list	A pointer to the list.
itr	The iterator after which to insert.
data	A pointer to the data to add. The memory pointed to by this parameter must be dynamically allocated, as an attempt will be made to `free()` it when deleting the list.

Returns

0 on success, CDSERR_BADITERATOR if `itr` is a NULL pointer.

4.6.2.9 int sl_list_insert_at (sl_list *list,* const size_t *index,* void ∗ *data*)

Parameters

list	A pointer to the list.
index	The index at which to insert. Setting this equal to the length of the list (i.e. to one element past the zero-based index of the last element) inserts the element at the end of the list.
data	A pointer to the data to add. The memory pointed to by this parameter must be dynamically allocated, as an attempt will be made to `free()` it when deleting the list.

Returns

0 on success, CDSERR_OUTOFRANGE if `index` exceeds the length of the list.

4.6.2.10 bool sl_list_isempty (const sl_list *list*)

Parameters

list	A pointer to the list.

Returns

`true` if the list is empty, otherwise `false`.

4.6.2.11 sl_list_itr sl_list_itr_from_index (const sl_list *list,* const size_t *index*)

Parameters

list	A pointer to the list.
index	The specified index.

Returns

The iterator, or NULL if `index` is out of range.

4.6.2.12 size_t sl_list_length (const sl_list *list*)

Parameters

list	A pointer to the list.

4.6.2.13 sl_list_itr sl_list_next (const sl_list_itr *itr*)

Parameters

itr	The iterator to advance

Returns

The advanced iterator.

4.6.2.14 void sl_list_prepend (sl_list *list*, void ∗ *data*)

Parameters

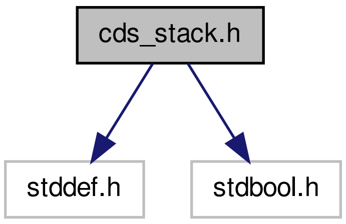
list	A pointer to the list.
data	A pointer to the data to add. The memory pointed to by this parameter must be dynamically allocated, as an attempt will be made to `free()` it when deleting the list.

4.7 cds_stack.h File Reference

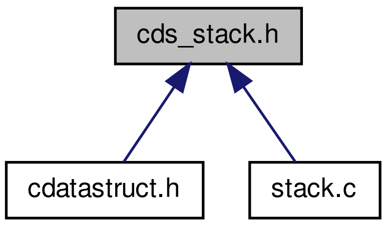
User interface to stack data structure.

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

Include dependency graph for cds_stack.h:



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



Typedefs

- typedef struct sl_list_t * stack

  Typedef for stack pointer.

Functions

- stack stack_init (void)

  Initializes a new stack.
- void stack_free (stack stk)

  Frees memory and releases resources used by a stack.
- size_t stack_length (const stack stk)

  Gets the number of items in a stack.
- bool stack_isempty (const stack stk)

  Checks if a stack is empty.
- void * stack_pop (stack stk)

  Pops a data item from the stack.
- void stack_push (stack stk, void *data)

  Pushes a data item onto the stack.

4.7.1 Detailed Description

Author

Paul Griffiths

Copyright

Copyright 2013 Paul Griffiths. Distributed under the terms of the GNU General Public License. http://www.gnu.org/licenses/

4.7.2 Function Documentation

4.7.2.1 void stack_free (stack *stk*)

Parameters

stk	A pointer to the stack.

4.7.2.2 stack stack_init (void)

Returns

A pointer to the new stack.

4.7.2.3 bool stack_isempty (const stack *stk*)

Parameters

stk	A pointer to the stack.

Returns

`true` is the stack is empty, `false` if not.

4.7.2.4 size_t stack_length (const stack *stk*)

Parameters

stk	A pointer to the stack.

Returns

The number of items in the stack.

4.7.2.5 void* stack_pop (stack *stk*)

The item returned was previously allocated using `malloc()`, so the user must `free()` the returned pointer when done.

Parameters

stk	A pointer to the stack.

Returns

A `void` pointer to the popped data item.

4.7.2.6 void stack_push (stack *stk,* void * *data*)

The provided pointer should point to dynamically allocated memory.

Parameters

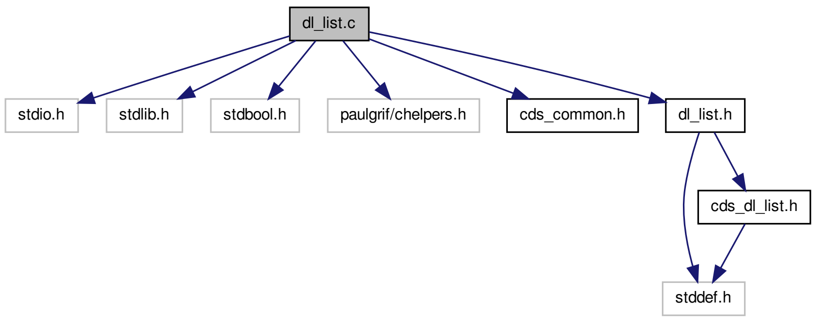
stk	A pointer to the stack.
data	A pointer to the data item to be pushed.

4.8 dl_list.c File Reference

Implementation of doubly linked list data structure.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <paulgrif/chelpers.h>
#include "cds_common.h"
#include "dl_list.h"
```

Include dependency graph for dl_list.c:



Functions

- dl_list dl_list_init (int(∗cfunc)(const void ∗, const void ∗))

  Initializes a new doubly linked list.
- void dl_list_free (dl_list list)

  Frees the resources associated with a list.
- size_t dl_list_length (const dl_list list)

  Returns the number of elements in a list.
- bool dl_list_isempty (const dl_list list)

  Checks if a list is empty.
- void dl_list_prepend (dl_list list, void ∗data)

  Inserts an element at the beginning of a list.
- void dl_list_append (dl_list list, void ∗data)

  Inserts an element at the end of a list.
- int dl_list_insert_before (dl_list list, const dl_list_itr itr, void ∗data)

  Inserts an element before a provided iterator.
- int dl_list_insert_at (dl_list list, const size_t index, void ∗data)

  Inserts an element at the specified index of a list.
- int dl_list_insert_after (dl_list list, const dl_list_itr itr, void ∗data)

  Inserts an element after a provided iterator.
- int dl_list_delete_at (dl_list list, const size_t index)

  Deletes a list element at a specified index.
- int dl_list_find_index (const dl_list list, const void ∗data)

  Finds the index of the specified data in a list.
- dl_list_itr dl_list_find_itr (const dl_list list, const void ∗data)

  Gets an iterator to the specified data in a list.

- void ∗ dl_list_data (const dl_list list, const size_t index)

  Returns a pointer to the data at a specified index.
- dl_list_itr dl_list_first (const dl_list list)

  Returns an iterator to the first element of a list.
- dl_list_itr dl_list_last (const dl_list list)

  Returns an iterator to the last element of a list.
- dl_list_itr dl_list_next (const dl_list_itr itr)

  Advances a list iterator by one element.
- dl_list_itr dl_list_prev (const dl_list_itr itr)

  Backs up a list iterator by one element.
- dl_list_itr dl_list_itr_from_index (const dl_list list, const size_t index)

  Return an iterator to a specified element of a list.
- dl_list_node dl_list_new_node (void ∗data)

  Creates a new list node.
- void dl_list_free_node (dl_list_node node)

  Frees resources for a node and any data.
- void dl_list_insert_node_front (dl_list list, dl_list_node node)

  Inserts a node at the front of a list.
- void dl_list_insert_node_before_mid (dl_list list, dl_list_itr itr, dl_list_node node)

  Inserts a node in the middle of a list before a specified iterator.
- void dl_list_insert_node_after_mid (dl_list list, dl_list_itr itr, dl_list_node node)

  Inserts a node in the middle of a list after a specified iterator.
- void dl_list_insert_node_back (dl_list list, dl_list_node node)

  Inserts a node at the back of a list.
- dl_list_node dl_list_remove_at (dl_list list, const size_t index)

  Removes, but does not delete, an element at an index.
- dl_list_node dl_list_remove_node_front (dl_list list)

  Removes the first node of a list.
- dl_list_node dl_list_remove_node_mid (dl_list list, dl_list_node node)

  Removes a specifed node from the middle of a list.
- dl_list_node dl_list_remove_node_back (dl_list list)

  Removes the last node of a list.
- void dl_list_find (const dl_list list, const void ∗data, dl_list_itr ∗p_itr, int ∗p_index)

  Finds the index of, and a pointer to, the first node in the list containing the specified data.

4.8.1 Detailed Description

Author

Paul Griffiths

Copyright

4.8.2 Function Documentation

4.8.2.1 void dl_list_append (dl_list *list*, void ∗ *data*)

Parameters

list	A pointer to the list.
data	A pointer to the data to add. The memory pointed to by this parameter must be dynamically allocated, as an attempt will be made to `free()` it when deleting the list.

4.8.2.2 void∗ dl_list_data (const dl_list *list,* const size_t *index*)

Parameters

list	A pointer to the list.
index	The index of the data.

Returns

A pointer to the data, or NULL if the index is out of range.

4.8.2.3 int dl_list_delete_at (dl_list *list,* const size_t *index*)

Parameters

list	A pointer to the list.
index	The index of the element to delete.

Returns

0 on success, CDSERR_OUTOFRANGE if the the index is out of range.

4.8.2.4 void dl_list_find (const dl_list *list,* const void ∗ *data,* dl_list_itr ∗ *p_itr,* int ∗ *p_index*)

Parameters

list	A pointer to the list.
data	A pointer to the data to find.
p_itr	A pointer to an iterator to populate with the result. This is set to CDSERR_NOTFOUND if the data was not found.
p_index	A pointer to an integer the populate with the result. This is set to NULL if the data was not found.

4.8.2.5 int dl_list_find_index (const dl_list *list,* const void ∗ *data*)

Parameters

list	A pointer to the list.
data	A pointer to the data to find.

Returns

The index of the element, if found, or CDSERR_NOTFOUND if it is not in the list.

4.8.2.6 dl_list_itr dl_list_find_itr (const dl_list *list,* const void ∗ *data*)

Parameters

list	A pointer to the list.
data	A pointer to the data to find.

Returns

An iterator to the found element, or NULL is the element is not in the list.

4.8.2.7 dl_list_itr dl_list_first (const dl_list *list*)

Parameters

list	A pointer to the list.

Returns

An iterator to the first element.

4.8.2.8 void dl_list_free (dl_list *list*)

Parameters

list	A pointer to the list to free.

4.8.2.9 void dl_list_free_node (dl_list_node *node*)

Parameters

node	A pointer to the node to free.

4.8.2.10 dl_list dl_list_init (int(∗)(const void ∗, const void ∗) *cfunc*)

Parameters

cfunc	A pointer to a compare function. The function should return `int` and accept two parameters of type `void *`. It should return less than 1 if the first parameter is less than the second, greater than 1 if the first parameter is greater than the second, and zero if the parameters are equal.

Returns

A pointer to the new list.

4.8.2.11 int dl_list_insert_after (dl_list *list,* const dl_list_itr *itr,* void ∗ *data*)

Note that dl_list_first() may return a NULL iterator when the list is empty. One reasonable behavior for this function would be to add a new node to the list in that case. However, an iterator may also become NULL when advanced to the end of the list. One possible way to modify this function would be to check the length of this list when the iterator is NULL, and if it is zero, add the first node to the list. However, the semantic meaning of adding an element *after* an iterator breaks down if that that iterator does not point to an existing element. Therefore, it is simpler for this function to simply refuse to handle NULL iterators. It is unlikely a user would want to call this function unless there are already elements in a list, and a valid iterator has been returned, e.g. through a find function.

Parameters

list	A pointer to the list.
itr	The iterator after which to insert.
data	A pointer to the data to add. The memory pointed to by this parameter must be dynamically allocated, as an attempt will be made to `free()` it when deleting the list.

Returns

0 on success, CDSERR_BADITERATOR if `itr` is a NULL pointer.

4.8.2.12 int dl_list_insert_at (**dl_list** *list,* const size_t *index,* void * *data*)

Parameters

list	A pointer to the list.
index	The index at which to insert. Setting this equal to the length of the list (i.e. to one element past the zero-based index of the last element) inserts the element at the end of the list.
data	A pointer to the data to add. The memory pointed to by this parameter must be dynamically allocated, as an attempt will be made to `free()` it when deleting the list.

Returns

0 on success, CDSERR_OUTOFRANGE if `index` exceeds the length of the list.

4.8.2.13 int dl_list_insert_before (**dl_list** *list,* const **dl_list_itr** *itr,* void * *data*)

Parameters

list	A pointer to the list.
itr	The iterator after which to insert.
data	A pointer to the data to add. The memory pointed to by this parameter must be dynamically allocated, as an attempt will be made to `free()` it when deleting the list.

Returns

0 on success, CDSERR_BADITERATOR if `itr` is a NULL pointer.

4.8.2.14 void dl_list_insert_node_after_mid (**dl_list** *list,* **dl_list_itr** *itr,* **dl_list_node** *node*)

Parameters

list	A pointer to the list.
itr	The iterator after which to insert. As this is inserting in the middle, this iterator should not be either the front or the back of the list, i.e. both the `prev` and `next` members should be non-NULL.
node	A pointer to the node to insert.

4.8.2.15 void dl_list_insert_node_back (**dl_list** *list,* **dl_list_node** *node*)

Parameters

list	A pointer to the list.
node	A pointer to the node to insert.

4.8.2.16 void dl_list_insert_node_before_mid (**dl_list** *list,* **dl_list_itr** *itr,* **dl_list_node** *node*)

Parameters

list	A pointer to the list.
itr	The iterator before which to insert. As this is inserting in the middle, this iterator should not be either the front or the back of the list, i.e. both the `prev` and `next` members should be non-NULL.
node	A pointer to the node to insert.

4.8.2.17 void dl_list_insert_node_front (**dl_list** *list,* **dl_list_node** *node*)

Parameters

list	A pointer to the list.
node	A pointer to the node to insert.

4.8.2.18 bool dl_list_isempty (const **dl_list** *list*)

Parameters

list	A pointer to the list.

Returns

`true` if the list is empty, otherwise `false`.

4.8.2.19 **dl_list_itr** dl_list_itr_from_index (const **dl_list** *list,* const size_t *index*)

Parameters

list	A pointer to the list.
index	The specified index.

Returns

The iterator, or NULL if `index` is out of range.

4.8.2.20 **dl_list_itr** dl_list_last (const **dl_list** *list*)

Parameters

list	A pointer to the list.

Returns

An iterator to the first element.

4.8.2.21 size_t dl_list_length (const **dl_list** *list*)

Parameters

list	A pointer to the list.

4.8.2.22 dl_list_node dl_list_new_node (void * *data*)

Parameters

data	The data for the new node.

Returns

A pointer to the newly created node.

4.8.2.23 dl_list_itr dl_list_next (const dl_list_itr *itr*)

Parameters

itr	The iterator to advance

Returns

The advanced iterator.

4.8.2.24 void dl_list_prepend (dl_list *list,* void * *data*)

Parameters

list	A pointer to the list.
data	A pointer to the data to add. The memory pointed to by this parameter must be dynamically allocated, as an attempt will be made to `free()` it when deleting the list.

4.8.2.25 dl_list_itr dl_list_prev (const dl_list_itr *itr*)

Parameters

itr	The iterator to back up.

Returns

The backed up iterator.

4.8.2.26 dl_list_node dl_list_remove_at (dl_list *list,* const size_t *index*)

Parameters

list	A pointer to the list.
index	The index of the element to be removed.

Returns

A pointer to the removed node. This should be `free()`d by calling dl_list_free_node().

4.8.2.27 dl_list_node dl_list_remove_node_back (dl_list *list*)

Parameters

list	A pointer to the list.

Returns

A pointer to the removed node.

4.8.2.28 dl_list_node dl_list_remove_node_front (dl_list *list*)

Parameters

list	A pointer to the list.

Returns

A pointer to the removed node.

4.8.2.29 dl_list_node dl_list_remove_node_mid (dl_list *list*, dl_list_node *node*)

Parameters

list	A pointer to the list.
node	A pointer to the node to remove. As this is removing from the middle, this node should not be either the front or the back of the list, i.e. both the `prev` and `next` members should be non-NULL.

Returns

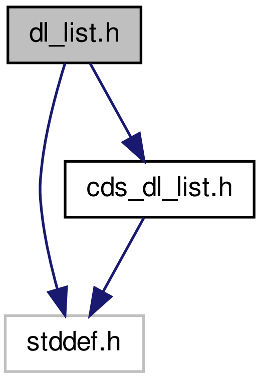
A pointer to the removed node, i.e. equal to `itr`.

4.9 dl_list.h File Reference

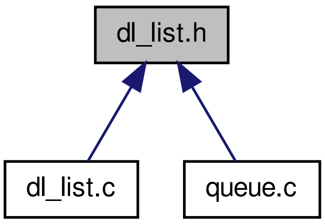
Developer interface to double linked list data structure.

```
#include <stddef.h>
#include "cds_dl_list.h"
```

Include dependency graph for dl_list.h:



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



Data Structures

- struct dl_list_t

    Struct to contain a list.

Typedefs

- typedef struct dl_list_t dl_list_t

    Struct to contain a list.
- typedef struct dl_list_node_t ∗ dl_list_node

    Typedef for list node.

Functions

- dl_list_node dl_list_new_node (void ∗data)

Creates a new list node.
- void dl_list_free_node (dl_list_node node)

  Frees resources for a node and any data.
- void dl_list_insert_node_front (dl_list list, dl_list_node node)

  Inserts a node at the front of a list.
- void dl_list_insert_node_before_mid (dl_list list, dl_list_itr itr, dl_list_node node)

  Inserts a node in the middle of a list before a specified iterator.
- void dl_list_insert_node_after_mid (dl_list list, dl_list_itr itr, dl_list_node node)

  Inserts a node in the middle of a list after a specified iterator.
- void dl_list_insert_node_back (dl_list list, dl_list_node node)

  Inserts a node at the back of a list.
- dl_list_node dl_list_remove_at (dl_list list, const size_t index)

  Removes, but does not delete, an element at an index.
- dl_list_node dl_list_remove_node_front (dl_list list)

  Removes the first node of a list.
- dl_list_node dl_list_remove_node_mid (dl_list list, dl_list_itr itr)

  Removes a specifed node from the middle of a list.
- dl_list_node dl_list_remove_node_back (dl_list list)

  Removes the last node of a list.
- void dl_list_find (const dl_list list, const void ∗data, dl_list_itr ∗p_itr, int ∗p_index)

  Finds the index of, and a pointer to, the first node in the list containing the specified data.

4.9.1 Detailed Description

Author

Paul Griffiths

Copyright

Copyright 2013 Paul Griffiths. Distributed under the terms of the GNU General Public License. `http://www.gnu.org/licenses/`

4.9.2 Function Documentation

4.9.2.1 void dl_list_find (const **dl_list** *list,* const void ∗ *data,* **dl_list_itr** ∗ *p_itr,* int ∗ *p_index*)

Parameters

list	A pointer to the list.
data	A pointer to the data to find.
p_itr	A pointer to an iterator to populate with the result. This is set to CDSERR_NOTFOUND if the data was not found.
p_index	A pointer to an integer the populate with the result. This is set to NULL if the data was not found.

4.9.2.2 void dl_list_free_node (**dl_list_node** *node*)

Parameters

node	A pointer to the node to free.

4.9.2.3 void dl_list_insert_node_after_mid (dl_list *list,* dl_list_itr *itr,* dl_list_node *node*)

Parameters

list	A pointer to the list.
itr	The iterator after which to insert. As this is inserting in the middle, this iterator should not be either the front or the back of the list, i.e. both the `prev` and `next` members should be non-NULL.
node	A pointer to the node to insert.

4.9.2.4 void dl_list_insert_node_back (dl_list *list,* dl_list_node *node*)

Parameters

list	A pointer to the list.
node	A pointer to the node to insert.

4.9.2.5 void dl_list_insert_node_before_mid (dl_list *list,* dl_list_itr *itr,* dl_list_node *node*)

Parameters

list	A pointer to the list.
itr	The iterator before which to insert. As this is inserting in the middle, this iterator should not be either the front or the back of the list, i.e. both the `prev` and `next` members should be non-NULL.
node	A pointer to the node to insert.

4.9.2.6 void dl_list_insert_node_front (dl_list *list,* dl_list_node *node*)

Parameters

list	A pointer to the list.
node	A pointer to the node to insert.

4.9.2.7 dl_list_node dl_list_new_node (void ∗ *data*)

Parameters

data	The data for the new node.

Returns

A pointer to the newly created node.

4.9.2.8 dl_list_node dl_list_remove_at (dl_list *list,* const size_t *index*)

Parameters

list	A pointer to the list.
index	The index of the element to be removed.

Returns

A pointer to the removed node. This should be `free()`d by calling dl_list_free_node().

4.9.2.9 dl_list_node dl_list_remove_node_back (dl_list *list*)

Parameters

list	A pointer to the list.

Returns

A pointer to the removed node.

4.9.2.10 dl_list_node dl_list_remove_node_front (dl_list *list*)

Parameters

list	A pointer to the list.

Returns

A pointer to the removed node.

4.9.2.11 dl_list_node dl_list_remove_node_mid (dl_list *list,* dl_list_node *node*)

Parameters

list	A pointer to the list.
node	A pointer to the node to remove. As this is removing from the middle, this node should not be either the front or the back of the list, i.e. both the `prev` and `next` members should be non-NULL.

Returns

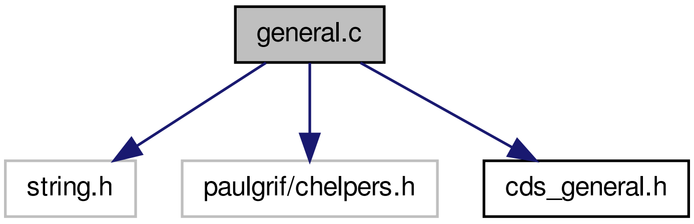
A pointer to the removed node, i.e. equal to `itr`.

4.10 general.c File Reference

Implementation of general data structure helper functions.

```
#include <string.h>
#include <paulgrif/chelpers.h>
#include "cds_general.h"
```

Include dependency graph for general.c:



Functions

- void ∗ cds_new_int (const int n)

  Dynamically allocates memory for a new `int`.
- void ∗ cds_new_uint (const unsigned int n)

  Dynamically allocates memory for a new `unsigned int`.
- void ∗ cds_new_long (const long n)

  Dynamically allocates memory for a new `long`.
- void ∗ cds_new_ulong (const unsigned long n)

  Dynamically allocates memory for a new `unsigned long`.
- void ∗ cds_new_string (const char ∗str)

  Dynamically allocates memory for a new string.
- int cds_compare_int (const void ∗data, const void ∗cmp)

  Compares two `int` via `void` pointers.
- int cds_compare_uint (const void ∗data, const void ∗cmp)

  Compares two `unsigned int` via `void` pointers.
- int cds_compare_long (const void ∗data, const void ∗cmp)

  Compares two `long` via `void` pointers.
- int cds_compare_ulong (const void ∗data, const void ∗cmp)

  Compares two `unsigned long` via `void` pointers.
- int cds_compare_string (const void ∗data, const void ∗cmp)

  Compares two strings via `void` pointers.

4.10.1 Detailed Description

Implementation of general data structure helper functions.

Author

Paul Griffiths

Copyright

4.10.2 Function Documentation

4.10.2.1 int cds_compare_int (const void ∗ *data*, const void ∗ *cmp*)

Parameters

data	Pointer to the data to which to compare.
cmp	Pointer to the comparison data.

Returns

-1 if the comparison data is greater than the data, 1 if the comparison data is less than the data, and 0 if the comparison data is equal to the data.

4.10.2.2 int cds_compare_long (const void ∗ *data*, const void ∗ *cmp*)

Parameters

data	Pointer to the data to which to compare.
cmp	Pointer to the comparison data.

Returns

-1 if the comparison data is greater than the data, 1 if the comparison data is less than the data, and 0 if the comparison data is equal to the data.

4.10.2.3 int cds_compare_string (const void ∗ *data*, const void ∗ *cmp*)

Parameters

data	Pointer to the data to which to compare.
cmp	Pointer to the comparison data.

Returns

-1 if the comparison data is greater than the data, 1 if the comparison data is less than the data, and 0 if the comparison data is equal to the data.

4.10.2.4 int cds_compare_uint (const void ∗ *data*, const void ∗ *cmp*)

Parameters

data	Pointer to the data to which to compare.
cmp	Pointer to the comparison data.

Returns

-1 if the comparison data is greater than the data, 1 if the comparison data is less than the data, and 0 if the comparison data is equal to the data.

4.10.2.5 int cds_compare_ulong (const void ∗ *data*, const void ∗ *cmp*)

Parameters

data	Pointer to the data to which to compare.
cmp	Pointer to the comparison data.

Returns

-1 if the comparison data is greater than the data, 1 if the comparison data is less than the data, and 0 if the comparison data is equal to the data.

4.10.2.6 void∗ cds_new_int (const int *n*)

Parameters

n	The new `int` for which to allocate.

Returns

A `void` pointer to the allocated memory.

4.10.2.7 void∗ cds_new_long (const long *n*)

Parameters

n	The new `long` for which to allocate.

Returns

A `void` pointer to the allocated memory.

4.10.2.8 void∗ cds_new_string (const char ∗ *str*)

Parameters

str	The new string for which to allocate.

Returns

A `void` pointer to the allocated memory.

4.10.2.9 void∗ cds_new_uint (const unsigned int *n*)

Parameters

n	The new `unsigned int` for which to allocate.

Returns

A `void` pointer to the allocated memory.

4.10.2.10 void∗ cds_new_ulong (const unsigned long *n*)

Parameters

n	The new `unsigned long` for which to allocate.

Returns

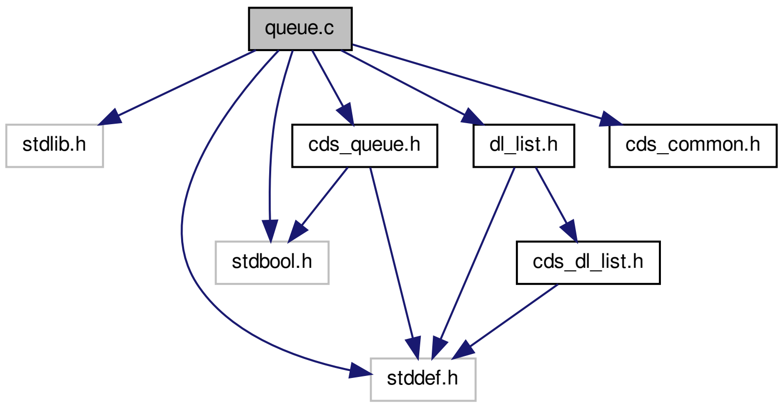
A `void` pointer to the allocated memory.

4.11 queue.c File Reference

Implementation of queue data structure.

```
#include <stdlib.h>
#include <stddef.h>
#include <stdbool.h>
#include "cds_queue.h"
#include "dl_list.h"
#include "cds_common.h"
```

Include dependency graph for queue.c:



Functions

- queue queue_init (void)

    Initializes a new queue.
- void queue_free (queue que)

    Frees memory and releases resources used by a queue.
- size_t queue_length (const queue que)

    Gets the number of items in a queue.
- bool queue_isempty (const queue que)

    Checks if a queue is empty.
- void ∗ queue_pop (queue que)

    Pops a data item from the queue.
- void queue_pushback (queue que, void ∗data)

    Pushes a data item onto the back of the queue.

4.11.1 Detailed Description

Implemented in terms of a doubly linked, double-ended list data structure.

Author

Paul Griffiths

Copyright

Copyright 2013 Paul Griffiths. Distributed under the terms of the GNU General Public License. `http://www.gnu.org/licenses/`

4.11.2 Function Documentation

4.11.2.1 void queue_free (queue *que*)

Parameters

que	A pointer to the queue.

4.11.2.2 queue queue_init (void)

Returns

A pointer to the new queue.

4.11.2.3 bool queue_isempty (const queue *que*)

Parameters

que	A pointer to the queue.

Returns

`true` is the queue is empty, `false` if not.

4.11.2.4 size_t queue_length (const queue *que*)

Parameters

que	A pointer to the queue.

Returns

The number of items in the queue.

4.11.2.5 void$*$ queue_pop (queue *que*)

The item returned was previously allocated using `malloc()`, so the user must `free()` the returned pointer when done.

Parameters

que	A pointer to the queue.

Returns

A `void` pointer to the popped data item.

4.11.2.6 void queue_pushback (queue *que*, void ∗ *data*)

The provided pointer should point to dynamically allocated memory.

Parameters

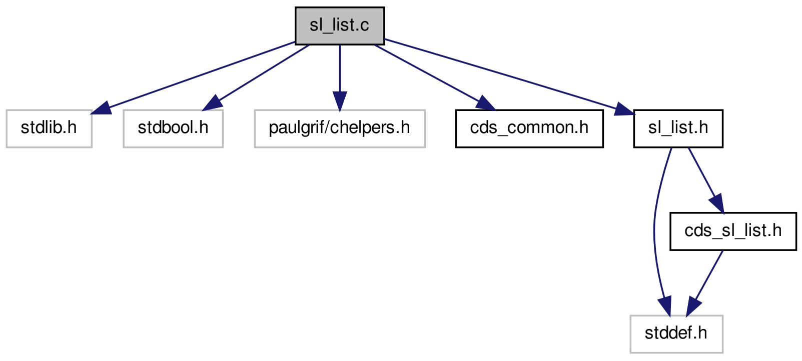
que	A pointer to the queue.
data	A pointer to the data item to be pushed.

4.12 sl_list.c File Reference

Implementation of singly linked list data structure.

```
#include <stdlib.h>
#include <stdbool.h>
#include <paulgrif/chelpers.h>
#include "cds_common.h"
#include "sl_list.h"
```

Include dependency graph for sl_list.c:



Functions

- sl_list sl_list_init (int(∗cfunc)(const void ∗, const void ∗))

  Initializes a new singly linked list.
- void sl_list_free (sl_list list)

  Frees the resources associated with a list.
- size_t sl_list_length (const sl_list list)

  Returns the number of elements in a list.
- bool sl_list_isempty (const sl_list list)

Checks if a list is empty.
- void sl_list_prepend (sl_list list, void ∗data)

  Inserts an element at the beginning of a list.
- int sl_list_insert_at (sl_list list, const size_t index, void ∗data)

  Inserts an element at the specified index of a list.
- int sl_list_insert_after (sl_list list, const sl_list_itr itr, void ∗data)

  Inserts an element after a provided iterator.
- int sl_list_delete_at (sl_list list, const size_t index)

  Deletes a list element at a specified index.
- int sl_list_find_index (const sl_list list, const void ∗data)

  Gets an index to the specified data in a list.
- sl_list_itr sl_list_find_itr (const sl_list list, const void ∗data)

  Gets an iterator to the specified data in a list.
- void ∗ sl_list_data (const sl_list list, const size_t index)

  Returns a pointer to the data at a specified index.
- sl_list_itr sl_list_first (const sl_list list)

  Returns an iterator to the first element of a list.
- sl_list_itr sl_list_next (const sl_list_itr itr)

  Advances a list iterator by one element.
- sl_list_itr sl_list_itr_from_index (const sl_list list, const size_t index)

  Return an iterator to a specified element of a list.
- sl_list_node sl_list_new_node (void ∗data)

  Creates a new list node.
- void sl_list_free_node (sl_list_node node)

  Frees resources for a node and any data.
- sl_list_node sl_list_remove_at (sl_list list, const size_t index)

  Removes, but does not delete, an element at an index.
- void sl_list_find (const sl_list list, const void ∗data, sl_list_itr ∗p_itr, int ∗p_index)

  Gets an index and iterator to a specified piece of data.

4.12.1 Detailed Description

Author

Paul Griffiths

Copyright

Copyright 2013 Paul Griffiths. Distributed under the terms of the GNU General Public License. http-://www.gnu.org/licenses/

4.12.2 Function Documentation

4.12.2.1 void∗ sl_list_data (const **sl_list** *list,* const size_t *index*)

Parameters

list	A pointer to the list.
index	The index of the data.

Returns

A pointer to the data, or NULL if the index is out of range.

4.12.2.2 int sl_list_delete_at (sl_list *list*, const size_t *index*)

Parameters

list	A pointer to the list.
index	The index of the element to delete.

Returns

0 on success, CDSERR_OUTOFRANGE if the the index is out of range.

4.12.2.3 void sl_list_find (const sl_list *list*, const void ∗ *data*, sl_list_itr ∗ *p_itr*, int ∗ *p_index*)

Parameters

list	A pointer to the list.
data	A pointer to the data to find.
p_itr	A pointer to an iterator to populate with the result. This parameter is ignored if set to NULL.
p_index	A pointer to an integer index to populate with the result. This parameter is ignored if set to NULL.

4.12.2.4 int sl_list_find_index (const sl_list *list*, const void ∗ *data*)

Parameters

list	A pointer to the list.
data	A pointer to the data to find.

Returns

The index of the found element, or CDSERR_NOTFOUND if the element is not in the list.

4.12.2.5 sl_list_itr sl_list_find_itr (const sl_list *list*, const void ∗ *data*)

Parameters

list	A pointer to the list.
data	A pointer to the data to find.

Returns

An iterator to the found element, or NULL is the element is not in the list.

4.12.2.6 sl_list_itr sl_list_first (const sl_list *list*)

Parameters

list	A pointer to the list.

Returns

An iterator to the first element.

4.12.2.7 void sl_list_free (sl_list *list*)

Parameters

list	A pointer to the list to free.

4.12.2.8 void sl_list_free_node (sl_list_node *node*)

Parameters

node	A pointer to the node to free.

4.12.2.9 sl_list sl_list_init (int(∗)(const void ∗, const void ∗) *cfunc*)

Parameters

cfunc	A pointer to a compare function. The function should return `int` and accept two parameters of type `void *`. It should return less than 1 if the first parameter is less than the second, greater than 1 if the first parameter is greater than the second, and zero if the parameters are equal.

Returns

A pointer to the new list.

4.12.2.10 int sl_list_insert_after (sl_list *list,* const sl_list_itr *itr,* void ∗ *data*)

Parameters

list	A pointer to the list.
itr	The iterator after which to insert.
data	A pointer to the data to add. The memory pointed to by this parameter must be dynamically allocated, as an attempt will be made to `free()` it when deleting the list.

Returns

0 on success, CDSERR_BADITERATOR if `itr` is a NULL pointer.

4.12.2.11 int sl_list_insert_at (sl_list *list,* const size_t *index,* void ∗ *data*)

Parameters

list	A pointer to the list.
index	The index at which to insert. Setting this equal to the length of the list (i.e. to one element past the zero-based index of the last element) inserts the element at the end of the list.
data	A pointer to the data to add. The memory pointed to by this parameter must be dynamically allocated, as an attempt will be made to `free()` it when deleting the list.

Returns

0 on success, CDSERR_OUTOFRANGE if `index` exceeds the length of the list.

4.12.2.12 bool sl_list_isempty (const **sl_list** *list*)

Parameters

list	A pointer to the list.

Returns

`true` if the list is empty, otherwise `false`.

4.12.2.13 **sl_list_itr** sl_list_itr_from_index (const **sl_list** *list,* const size_t *index*)

Parameters

list	A pointer to the list.
index	The specified index.

Returns

The iterator, or NULL if `index` is out of range.

4.12.2.14 size_t sl_list_length (const **sl_list** *list*)

Parameters

list	A pointer to the list.

4.12.2.15 **sl_list_node** sl_list_new_node (void ∗ *data*)

Parameters

data	The data for the new node.

Returns

A pointer to the newly created node.

4.12.2.16 **sl_list_itr** sl_list_next (const **sl_list_itr** *itr*)

Parameters

itr	The iterator to advance

Returns

The advanced iterator.

4.12.2.17 void sl_list_prepend (**sl_list** *list,* void ∗ *data*)

Parameters

list	A pointer to the list.
data	A pointer to the data to add. The memory pointed to by this parameter must be dynamically allocated, as an attempt will be made to `free()` it when deleting the list.

4.12.2.18 sl_list_node sl_list_remove_at (sl_list *list*, const size_t *index*)

Parameters

list	A pointer to the list.
index	The index of the element to be removed.

Returns

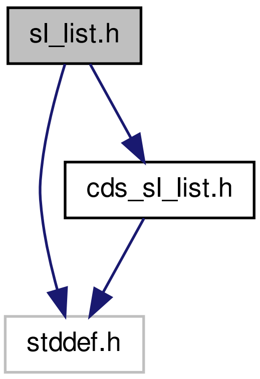
A pointer to the removed node. This should be `free()`d by calling sl_list_free_node().

4.13 sl_list.h File Reference

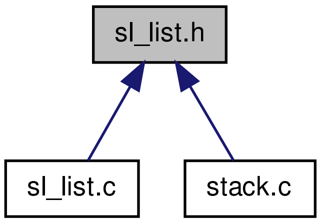
Developer interface to singly linked list data structure.

```
#include <stddef.h>
#include "cds_sl_list.h"
```

Include dependency graph for sl_list.h:



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



Data Structures

- struct sl_list_t

    Struct to contain a list.

Typedefs

- typedef struct sl_list_t sl_list_t

    Struct to contain a list.
- typedef struct sl_list_node_t ∗ sl_list_node

    Typedef for list node.

Functions

- sl_list_node sl_list_new_node (void ∗data)

    Creates a new list node.
- void sl_list_free_node (sl_list_node node)

    Frees resources for a node and any data.
- sl_list_node sl_list_remove_at (sl_list list, const size_t index)

    Removes, but does not delete, an element at an index.
- void sl_list_find (const sl_list list, const void ∗data, sl_list_itr ∗p_itr, int ∗p_index)

    Gets an index and iterator to a specified piece of data.

4.13.1 Detailed Description

Author

Paul Griffiths

Copyright

Copyright 2013 Paul Griffiths. Distributed under the terms of the GNU General Public License. `http://www.gnu.org/licenses/`

4.13.2 Function Documentation

4.13.2.1 void sl_list_find (const **sl_list** *list,* const void ∗ *data,* **sl_list_itr** ∗ *p_itr,* int ∗ *p_index*)

Parameters

list	A pointer to the list.
data	A pointer to the data to find.
p_itr	A pointer to an iterator to populate with the result. This parameter is ignored if set to NULL.
p_index	A pointer to an integer index to populate with the result. This parameter is ignored if set to NULL.

4.13.2.2 void sl_list_free_node (**sl_list_node** *node*)

Parameters

node	A pointer to the node to free.

4.13.2.3 **sl_list_node** sl_list_new_node (void ∗ *data*)

Parameters

data	The data for the new node.

Returns

A pointer to the newly created node.

4.13.2.4 **sl_list_node** sl_list_remove_at (**sl_list** *list,* const size_t *index*)

Parameters

list	A pointer to the list.
index	The index of the element to be removed.

Returns

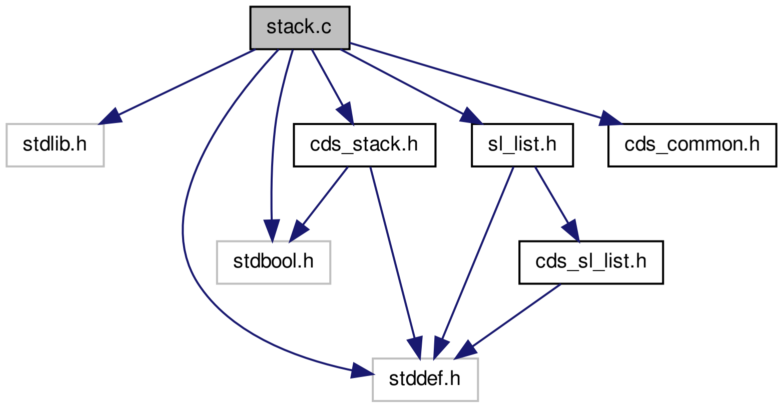
A pointer to the removed node. This should be `free()`d by calling sl_list_free_node().

4.14 stack.c File Reference

Implementation of stack data structure.

```
#include <stdlib.h>
#include <stddef.h>
#include <stdbool.h>
#include "cds_stack.h"
#include "sl_list.h"
#include "cds_common.h"
```

Include dependency graph for stack.c:



Functions

- stack stack_init (void)

    Initializes a new stack.
- void stack_free (stack stk)

    Frees memory and releases resources used by a stack.
- size_t stack_length (const stack stk)

    Gets the number of items in a stack.
- bool stack_isempty (const stack stk)

    Checks if a stack is empty.
- void ∗ stack_pop (stack stk)

    Pops a data item from the stack.
- void stack_push (stack stk, void ∗data)

    Pushes a data item onto the stack.

4.14.1 Detailed Description

Implemented in terms of a singly linked, singled-ended list data structure.

Author

Paul Griffiths

Copyright

Copyright 2013 Paul Griffiths. Distributed under the terms of the GNU General Public License. http-://www.gnu.org/licenses/

4.14.2 Function Documentation

4.14.2.1 void stack_free (stack *stk*)

Parameters

stk	A pointer to the stack.

4.14.2.2 stack stack_init (void)

Returns

A pointer to the new stack.

4.14.2.3 bool stack_isempty (const stack *stk*)

Parameters

stk	A pointer to the stack.

Returns

`true` is the stack is empty, `false` if not.

4.14.2.4 size_t stack_length (const stack *stk*)

Parameters

stk	A pointer to the stack.

Returns

The number of items in the stack.

4.14.2.5 void* stack_pop (stack *stk*)

The item returned was previously allocated using `malloc()`, so the user must `free()` the returned pointer when done.

Parameters

stk	A pointer to the stack.

Returns

A `void` pointer to the popped data item.

4.14.2.6 void stack_push (stack *stk*, void * *data*)

The provided pointer should point to dynamically allocated memory.

Parameters

stk	A pointer to the stack.
data	A pointer to the data item to be pushed.

Index