DGCC

Generated by Doxygen 1.8.17

Chapter 1

DGCC

1.1 Dynamic Generic C Collection

The project is a library regrouping several generic collections :

- Stack : First In First Out (FIFO)
- List
- OrderedList : TODO
- Queue : TODO
- PriorityQueue : TODO

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Class Documentation

4.1 List Struct Reference

Data structure for collection of element.

`#include <list.h>`

4.1.1 Detailed Description

Data structure for collection of element.

Note

0 is the first element

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

- Collection/list.h

4.2 Stack Struct Reference

Data structure for First In First Out (FIFO)

`#include <stack.h>`

4.2.1 Detailed Description

Data structure for First In First Out (FIFO)

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

- Collection/stack.h

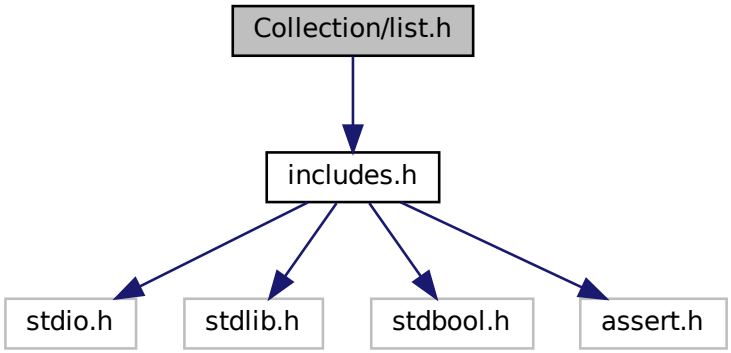
Chapter 5

File Documentation

5.1 Collection/list.h File Reference

```
#include "includes.h"
```

Include dependency graph for list.h:



Typedefs

- typedef struct s_List * **List**

Functions

- List list_new (void)

  Create a list empty.
- void list_destruct (List l)

  Destruct the list.
- unsigned int list_size (const List l)

Get the size of the list.

- bool list_isEmpty (const List l)

  Know if a list is empty.

- List list_add (List l, void *e)

  Add an element in the list.

- List list_remove (List l, const unsigned int index)

  Remove element from the list.

- List list_insert (List l, const unsigned int index, void *e)

  Insert an element at index in list.

- void * list_get (const List l, const unsigned int index)

  Obtain element at index from list.

- List list_map (List l, void *f(void *e))

  Apply the function f on the list.

- List list_sublist (const List l, const unsigned int index1, const unsigned int index2)

  Create a view/sublist with a list.

- bool list_exists (const List l, bool predicate(void *e))

  Check if it exists an element checking a predicate.

- bool list_forall (const List l, bool predicate(void *e))

  Check if all elements check the predicat.

5.1.1 Detailed Description

Author

Jessy Khafif

5.1.2 Function Documentation

5.1.2.1 list_add()

```
List list_add (
            List l,
            void * e )
```

Add an element in the list.

Parameters

in,out	*l*	: list
in	*e*	: generic element

Returns

list modified

Precondition

(l != NULL) and (e != NULL)

5.1.2.2 list_destruct()

```
void list_destruct (
            List l )
```

Destruct the list.

Parameters

in,out	*l*	: a list

Precondition

(l != NULL)

Postcondition

(l == NULL)

5.1.2.3 list_exists()

```
bool list_exists (
            const List l,
            bool  predicatevoid *e )
```

Check if it exists an element checking a predicate.

Parameters

in	*l*	: list
in	*predicate*	: predicate returning 0 if false

Returns

0 if it doesn't exist an element e such as predicate(e) == 0

5.1.2.4 list_forall()

```
bool list_forall (
```

```
            const List l,
            bool  predicatevoid *e )
```

Check if all elements check the predicat.

Parameters

in	*l*	: list
in	*predicate*	: predicate returning 0 if false

Returns

0 if it exists an element e such as predicate(e) == 0

5.1.2.5 list_get()

```
void* list_get (
            const List l,
            const unsigned int index )
```

Obtain element at index from list.

Parameters

in	*l*	: list
in	*index*	: position

Returns

: element

Precondition

(l != NULL) && (!list_isEmpty(l)) && (index < list_size(l))

5.1.2.6 list_insert()

```
List list_insert (
            List l,
            const unsigned int index,
            void * e )
```

Insert an element at index in list.

Note

list_add(l,e) <=> list_insert(l,list_size(l),e)

Parameters

in,out	*l*	: list
in	*index*	: position
in	*e*	: generic element

Returns

list modified

Precondition

(l != NULL) && (e != NULL) && (index < list_size(l))

5.1.2.7 list_isEmpty()

```
bool list_isEmpty (
            const List l )
```

Know if a list is empty.

Parameters

in	*l*	: list

Returns

list_isEmpty(l) == 1

Precondition

(l != NULL)

5.1.2.8 list_map()

```
List list_map (
            List l,
            void * fvoid *e )
```

Apply the function f on the list.

Parameters

in,out	*l*	: list
in	*f*	: function applied on the elements

Returns

list modified by f

Precondition

(l != NULL)

5.1.2.9 list_new()

```
List list_new (
            void  )
```

Create a list empty.

Returns

a list

Postcondition

(list_new() != NULL) && (list_isEmpty(list_new()) == 1)

5.1.2.10 list_remove()

```
List list_remove (
            List l,
            const unsigned int index )
```

Remove element from the list.

Parameters

in,out	*l*	: list
in	*index*	: index of element

Returns

list modified

Precondition

(l != NULL) && (index < list_size(l))

5.1.2.11 list_size()

```
unsigned int list_size (
            const List l )
```

Get the size of the list.

Parameters

in	*l*	: list

Returns

the lenght of list

Precondition

(l != NULL)

5.1.2.12 list_sublist()

```
List list_sublist (
            const List l,
            const unsigned int index1,
            const unsigned int index2 )
```

Create a view/sublist with a list.

Parameters

in	*l*	: list
in	*index1*	: the first position
in	*index2*	: the second position

Returns

the sublist

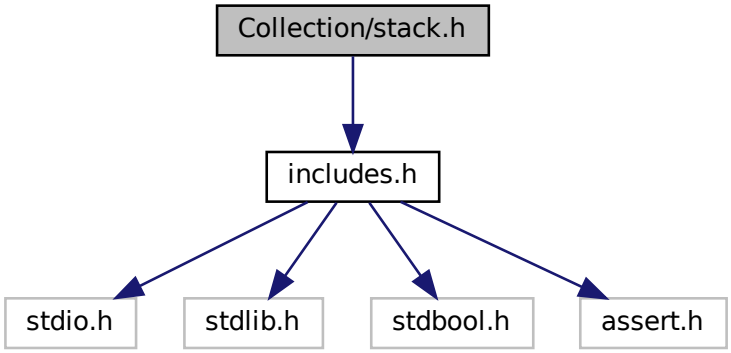
Precondition

(l != NULL) && (0 <= index1) && (index1 <= index2) && (index2 < list_size(l))

5.2 Collection/stack.h File Reference

```
#include "includes.h"
```

Include dependency graph for stack.h:



Typedefs

- typedef struct s_Stack ∗ **Stack**

Functions

- Stack stack_new (void)

  Create new stack.
- Stack stack_push (Stack s, void ∗e)

  Add element in stack.
- Stack stack_pop (Stack s)

  Remove element in stack.
- void ∗ stack_top (const Stack s)

  Get the element from the top of stack.
- int stack_size (const Stack s)

  Get the size of stack.
- bool stack_isEmpty (const Stack s)

  Know if the stack is empty.
- void stack_destruct (Stack s)

  Delete the stack.
- void stack_map (Stack s, void ∗f(void ∗e))

  Apply the function f on the stack.

5.2.1 Detailed Description

Author

Jessy Khafif

5.2.2 Function Documentation

5.2.2.1 stack_destruct()

```
void stack_destruct (
            Stack s )
```

Delete the stack.

Parameters

in,out	s	: stack

Precondition

(s != NULL)

Postcondition

(s == NULL)

5.2.2.2 stack_isEmpty()

```
bool stack_isEmpty (
            const Stack s )
```

Know if the stack is empty.

Parameters

in,out	s	: stack

Precondition

(s != NULL)

Returns

Stack is empty ?

5.2.2.3 stack_map()

```
void stack_map (
            Stack s,
            void * fvoid *e )
```

Apply the function f on the stack.

Parameters

in,out	*s*	: stack
in	*f*	: function

5.2.2.4 stack_new()

```
Stack stack_new (
            void  )
```

Create new stack.

Postcondition

stack_isEmpty(stack_new()) == true

Returns

stack

5.2.2.5 stack_pop()

```
Stack stack_pop (
            Stack s )
```

Remove element in stack.

Parameters

in,out	*s*	: stack

Precondition

(s != NULL) && (stack_isEmpty(s) == 0)

Returns

stack modified

5.2.2.6 stack_push()

```
Stack stack_push (
            Stack s,
            void * e )
```

Add element in stack.

Parameters

in,out	*s*	: stack
in	*e*	: generic element

Precondition

(s != NULL) && (e != NULL)

Returns

stack modified

5.2.2.7 stack_size()

```
int stack_size (
            const Stack s )
```

Get the size of stack.

Parameters

in,out	*s*	: stack

Precondition

(s != NULL)

Returns

size of stack

5.2.2.8 stack_top()

```
void* stack_top (
            const Stack s )
```

Get the element from the top of stack.

Parameters

in,out	*s*	: stack

Precondition

(s != NULL) && (stack_isEmpty(s) == 0)

Returns

the top element of stack

Index