

DGCC

Generated by Doxygen 1.8.17

1 DGCC	1
1.1 Dynamic Generic C Collection	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Class Documentation	7
4.1 List Struct Reference	7
4.1.1 Detailed Description	7
4.2 Stack Struct Reference	7
4.2.1 Detailed Description	7
5 File Documentation	9
5.1 Collection/list.h File Reference	9
5.1.1 Detailed Description	10
5.1.2 Function Documentation	10
5.1.2.1 list_add()	10
5.1.2.2 list_destruct()	11
5.1.2.3 list_exists()	11
5.1.2.4 list_forall()	11
5.1.2.5 list_get()	12
5.1.2.6 list_insert()	12
5.1.2.7 list_isEmpty()	13
5.1.2.8 list_map()	13
5.1.2.9 list_new()	14
5.1.2.10 list_remove()	14
5.1.2.11 list_size()	15
5.1.2.12 list_sublist()	15
5.2 Collection/stack.h File Reference	15
5.2.1 Detailed Description	16
5.2.2 Function Documentation	17
5.2.2.1 stack_destruct()	17
5.2.2.2 stack_isEmpty()	17
5.2.2.3 stack_map()	18
5.2.2.4 stack_new()	18
5.2.2.5 stack_pop()	18
5.2.2.6 stack_push()	19
5.2.2.7 stack_size()	19
5.2.2.8 stack_top()	20
Index	21

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:

List	Data structure for collection of element	7
Stack	Data structure for First In First Out (FIFO)	7

Chapter 3

File Index

3.1 File List

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

Collection/ includes.h	??
Collection/ list.h	9
Collection/ stack.h	15

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](#)

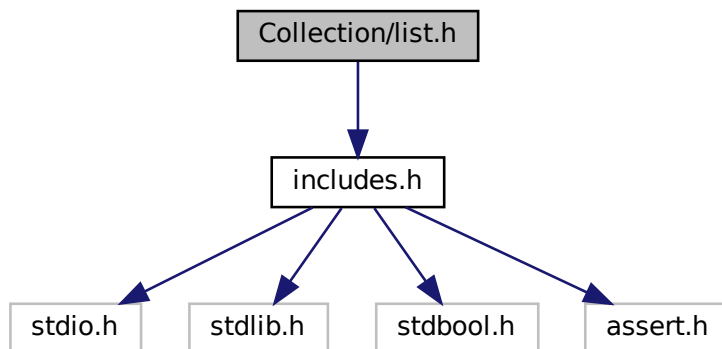
Chapter 5

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	<i>l</i>	: list
in	<i>e</i>	: 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	/	: 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	/	: list
in	<i>predicate</i>	: 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	<i>l</i>	: list
in	<i>predicate</i>	: 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	<i>l</i>	: list
in	<i>index</i>	: 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

<i>in, out</i>	<i>l</i>	: list
<i>in</i>	<i>index</i>	: position
<i>in</i>	<i>e</i>	: 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

<i>in</i>	<i>l</i>	: 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

<i>in, out</i>	<i>l</i>	: list
<i>in</i>	<i>f</i>	: 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	<i>l</i>	: list
in	<i>index</i>	: 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	/	: 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	/	: list
in	index1	: the first position
in	index2	: the second position

Returns

the sublist

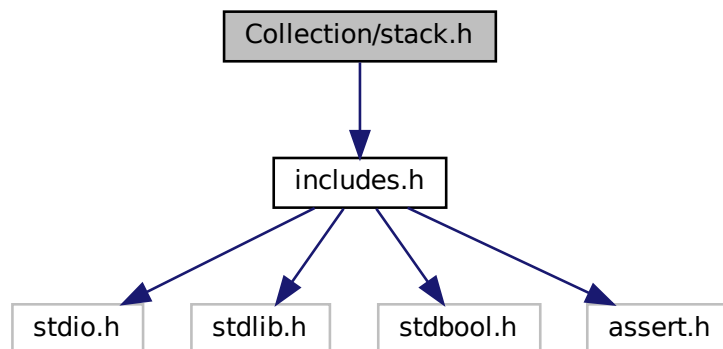
Precondition

(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

Collection/list.h, 9
Collection/stack.h, 15

List, 7

list.h

- list_add, 10
- list_destruct, 11
- list_exists, 11
- list_forall, 11
- list_get, 12
- list_insert, 12
- list_isEmpty, 13
- list_map, 13
- list_new, 14
- list_remove, 14
- list_size, 14
- list_sublist, 15

list_add

- list.h, 10

list_destruct

- list.h, 11

list_exists

- list.h, 11

list_forall

- list.h, 11

list_get

- list.h, 12

list_insert

- list.h, 12

list_isEmpty

- list.h, 13

list_map

- list.h, 13

list_new

- list.h, 14

list_remove

- list.h, 14

list_size

- list.h, 14

list_sublist

- list.h, 15

Stack, 7

stack.h

- stack_destruct, 17

- stack_isEmpty, 17

- stack_map, 17

- stack_new, 18

- stack_pop, 18

- stack_push, 19

- stack_size, 19

- stack_top, 19

stack_destruct

- stack.h, 17

stack_isEmpty

- stack.h, 17

stack_map

- stack.h, 17

stack_new

- stack.h, 18

stack_pop

- stack.h, 18

stack_push

- stack.h, 19

stack_size

- stack.h, 19

stack_top

- stack.h, 19