C Data Structures

0.1

Generated by Doxygen 1.8.20

Mon Nov 2 2020 23:10:59

1 Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

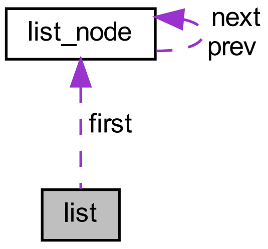
3 Data Structure Documentation

3.1 list Struct Reference

Structure for a list contaning le size and first element of the list.

```
#include <list.h>
```

Collaboration diagram for list:



Data Fields

- unsigned size
- struct list_node ∗ first

3.1.1 Detailed Description

Structure for a list contaning le size and first element of the list.

3.1.2 Field Documentation

3.1.2.1 first `struct list_node* list::first`

pointer to the first node of the list

3.1.2.2 size `unsigned list::size`

size of the list

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

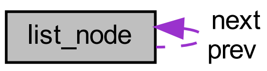
- include/data-structures/list.h

3.2 list_node Struct Reference

Structure for a list node contaning the value and the next/prev node pointer.

```
#include <list.h>
```

Collaboration diagram for list_node:



Data Fields

- void ∗ val
- struct list_node ∗ next
- struct list_node ∗ prev

3.2.1 Detailed Description

Structure for a list node contaning the value and the next/prev node pointer.

3.2.2 Field Documentation

3.2.2.1 next `struct list_node* list_node::next`

pointer to the next node

3.2.2.2 prev `struct list_node* list_node::prev`

pointer to the previous node

3.2.2.3 val `void* list_node::val`

pointer to the value

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

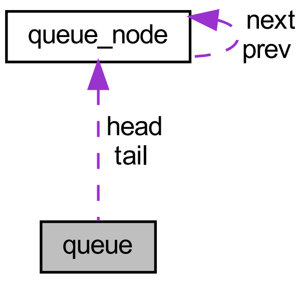
- include/data-structures/list.h

3.3 queue Struct Reference

Structure for a queue contaning le size, head and tail element of the queue.

`#include <queue.h>`

Collaboration diagram for queue:



Data Fields

- unsigned size
- struct queue_node * head
- struct queue_node * tail

3.3.1 Detailed Description

Structure for a queue contaning le size, head and tail element of the queue.

3.3.2 Field Documentation

3.3.2.1 head `struct queue_node* queue::head`

pointer to the head node of the queue

3.3.2.2 size `unsigned queue::size`

size of the queue

3.3.2.3 tail `struct queue_node* queue::tail`

pointer to the tail node of the queue

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

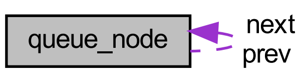
- include/data-structures/queue.h

3.4 queue_node Struct Reference

Structure for a queue_node contaning the value and the next/prev node pointer.

```
#include <queue.h>
```

Collaboration diagram for queue_node:



Data Fields

- void ∗ val
- struct queue_node ∗ next
- struct queue_node ∗ prev

3.4.1 Detailed Description

Structure for a queue_node contaning the value and the next/prev node pointer.

3.4.2 Field Documentation

3.4.2.1 next `struct queue_node* queue_node::next`

pointer to the next node

3.4.2.2 prev `struct queue_node* queue_node::prev`

pointer to the previous node

3.4.2.3 val `void* queue_node::val`

pointer to the value

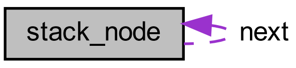
The documentation for this struct was generated from the following file:

- include/data-structures/queue.h

3.5 stack_node Struct Reference

```
#include <stack.h>
```

Collaboration diagram for stack_node:



Data Fields

- void ∗ val
- struct stack_node ∗ next

3.5.1 Detailed Description

Structure for a stack node contaning the value and the next node pointer.

3.5.2 Field Documentation

3.5.2.1 next `struct stack_node* stack_node::next`

pointer to the next node

3.5.2.2 val `void* stack_node::val`

pointer to the value

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

- include/data-structures/stack.h

3.6 test_rate Struct Reference

```
#include <test-suite.h>
```

Data Fields

- unsigned number_test
- unsigned number_success

3.6.1 Field Documentation

3.6.1.1 number_success `unsigned test_rate::number_success`

3.6.1.2 number_test `unsigned test_rate::number_test`

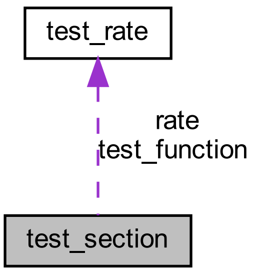
The documentation for this struct was generated from the following file:

- test/test-suite.h

3.7 test_section Struct Reference

```
#include <test-suite.h>
```

Collaboration diagram for test_section:



Data Fields

- char ∗ name
- struct test_rate(∗ test_function)(void)
- struct test_rate rate

3.7.1 Field Documentation

3.7.1.1 name `char* test_section::name`

3.7.1.2 rate `struct test_rate test_section::rate`

3.7.1.3 test_function `struct test_rate(* test_section::test_function) (void)`

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

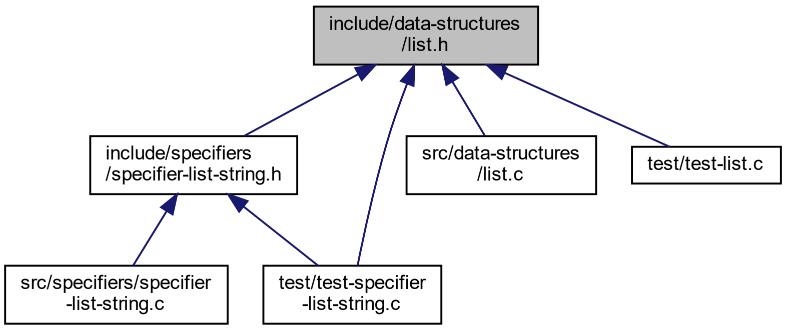
- test/test-suite.h

4 File Documentation

4.1 include/data-structures/list.h File Reference

List data structure file.

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


include/data-structures /list.h
include/specifiers /specifier-list-string.h
src/data-structures /list.c
test/test-list.c
src/specifiers/specifier -list-string.c
test/test-specifier -list-string.c

Data Structures

- struct list_node

    Structure for a list node contaning the value and the next/prev node pointer.
- struct list

    Structure for a list contaning le size and first element of the list.

Functions

- void list_init (struct list ∗list)

    *Initiate the **list** structure with default values*
    No allocation is done here
- unsigned list_size (struct list ∗list)

    Get the size of the list.
- int list_isEmpty (struct list ∗list)

    Check whether the list is empty.
- void list_append (struct list ∗list, struct list_node ∗node)

    Append a node to a list.
- void list_prepend (struct list ∗list, struct list_node ∗node)

Prepend a node to a list.
- struct list_node ∗ list_popleft (struct list ∗list)

  Pop the left node of a list
  No free is done here
- struct list_node ∗ list_popright (struct list ∗list)

  Pop the right node of a list
  No free is done here
- struct list_node ∗ list_get (struct list ∗list, unsigned index)

  Get a node from a list.
- int list_insert (struct list ∗list, struct list_node ∗node, unsigned index)

  Insert a node at an index of a list.
- struct list_node ∗ list_remove (struct list ∗list, unsigned index)

  Remove a node from a list
  No free is done here

4.1.1 Detailed Description

List data structure file.

Author

Sébastien Goubeau (`sebastien.goubeau@protonmail.com`)

Containes all the functions and structures needed to manipulate the list data structure.

4.1.2 Function Documentation

4.1.2.1 list_append()

```
void list_append (
        struct list * list,
        struct list_node * node )
```

Append a node to a list.

Parameters

list	List to append from
node	Node to append

4.1.2.2 list_get()

```
struct list_node* list_get (
        struct list * list,
        unsigned index )
```

Get a node from a list.

Parameters

list	List from which to get the node from
index	Index of the node to fetch from the left

Returns

Node's pointer

4.1.2.3 list_init()

```
void list_init (
        struct list * list )
```

Initiate the **list** structure with default values
No allocation is done here

Parameters

list	List structure

4.1.2.4 list_insert()

```
int list_insert (
        struct list * list,
        struct list_node * node,
        unsigned index )
```

Insert a node at an index of a list.

Parameters

list	List from which to insert to
node	Node to insert
index	Index of the new node in the **list**

Returns

true (1) if the node has been inserted, false (0) otherwise. The function returns false if the index is greater than the list's size.

4.1.2.5 list_isEmpty()

```
int list_isEmpty (
        struct list * list )
```

Check whether the list is empty.

Parameters

list	List of which to check emptiness

Returns

true (1) if the **list** is empty, false (0) otherwise

4.1.2.6 list_popleft()

```
struct list_node* list_popleft (
        struct list * list )
```

Pop the left node of a list
No free is done here

Parameters

list	List from which to pop from

Returns

The node that just been poped

4.1.2.7 list_popright()

```
struct list_node* list_popright (
        struct list * list )
```

Pop the right node of a list
No free is done here

Parameters

list	List from which to pop from

Returns

The node that just been poped

4.1.2.8 list_prepend()

```
void list_prepend (
        struct list * list,
        struct list_node * node )
```

Prepend a node to a list.

Parameters

list	List to prepend from
node	Node to prepend

4.1.2.9 list_remove()

```
struct list_node* list_remove (
        struct list * list,
        unsigned index )
```

Remove a node from a list
No free is done here

Parameters

list	List from which to remove from
index	Index of the node to remove

Returns

The node that just been removed. The function returns NULL if the index is greater or equal to the list's size.

4.1.2.10 list_size()

```
unsigned list_size (
        struct list * list )
```

Get the size of the list.

Parameters

list	List of which to get the size from

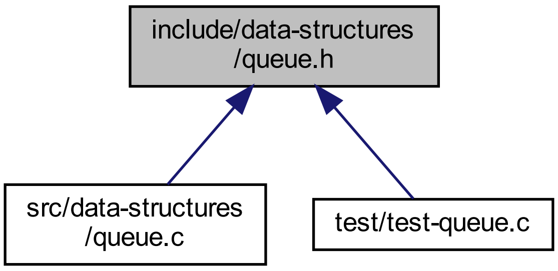
Returns

Size of the list **list**

4.2 include/data-structures/queue.h File Reference

Queue data structure file.

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



Data Structures

- struct queue_node

    Structure for a queue_node contaning the value and the next/prev node pointer.
- struct queue

    Structure for a queue contaning le size, head and tail element of the queue.

Functions

- void queue_init (struct queue ∗q)

    *Initiate the **queue** structure with default values*
    No allocation is done here
- unsigned queue_size (struct queue ∗q)

    Get the size of the queue.
- int queue_isEmpty (struct queue ∗q)

    Check whether the queue is empty.
- void queue_enqueue (struct queue ∗q, struct queue_node ∗n)

    Enqueue a node to a queue.
- struct queue_node ∗ queue_dequeue (struct queue ∗q)

    Dequeue a node from a queue
    No free is done here

4.2.1 Detailed Description

Queue data structure file.

Author

Sébastien Goubeau (`sebastien.goubeau@protonmail.com`)

Containes all the functions and structures needed to manipulate the queue data structure.

4.2.2 Function Documentation

4.2.2.1 queue_dequeue()

```
struct queue_node* queue_dequeue (
        struct queue * q )
```

Dequeue a node from a queue
No free is done here

Parameters

q	Queue to dequeue from

Returns

The node that just been dequeued. Returns NULL if the queue is empty.

4.2.2.2 queue_enqueue()

```
void queue_enqueue (
        struct queue * q,
        struct queue_node * n )
```

Enqueue a node to a queue.

Parameters

q	Queue to enqueue from
n	Node to enqueue

4.2.2.3 queue_init()

```
void queue_init (
        struct queue * q )
```

Initiate the **queue** structure with default values
No allocation is done here

Parameters

q	Queue structure

4.2.2.4 queue_isEmpty()

```
int queue_isEmpty (
        struct queue * q )
```

Check whether the queue is empty.

Parameters

q	Queue of which to check emptiness

Returns

true (1) if the **queue** is empty, false (0) otherwise

4.2.2.5 queue_size()

```
unsigned queue_size (
        struct queue * q )
```

Get the size of the queue.

Parameters

q	Queue of which to get the size from

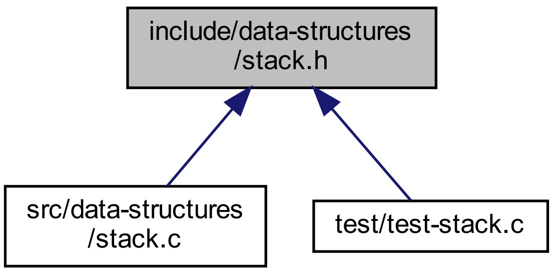
Returns

Size of the queue **queue**

4.3 include/data-structures/stack.h File Reference

Stack data structure file.

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


include/data-structures /stack.h
src/data-structures /stack.c
test/test-stack.c

Data Structures

- struct stack_node

Functions

- void stack_initStack (struct stack_node ∗stack)

  *Initiate the **stack** structure with default vales*
  No allocation is done here
- int stack_isStackEmpty (struct stack_node ∗stack)

  Check whether the stack is empty.
- void stack_push (struct stack_node ∗stack, struct stack_node ∗node)

  Push a node on top of the stack.
- struct stack_node ∗ stack_pop (struct stack_node ∗stack)

  Pop the node on top of the stack
  No free is done here

4.3.1 Detailed Description

Stack data structure file.

Author

Sébastien Goubeau (`sebastien.goubeau@protonmail.com`)

Containes all the functions and structures needed to manipulate the stack data structure.

4.3.2 Function Documentation

4.3.2.1 stack_initStack()

```
void stack_initStack (
    struct stack_node * stack )
```

Initiate the **stack** structure with default vales
No allocation is done here

Parameters

stack	Top node of the stack

The first node of the stack can be considered the stack it self in this implementation.

4.3.2.2 stack_isStackEmpty()

```
int stack_isStackEmpty (
    struct stack_node * stack )
```

Check whether the stack is empty.

Parameters

stack	Top node of the stack

Returns

true (1) if the **stack** is empty, false (0) otherwise

4.3.2.3 stack_pop()

```
struct stack_node* stack_pop (
        struct stack_node * stack )
```

Pop the node on top of the stack
No free is done here

Parameters

stack	Top node of the stack

Returns

The node that just been poped

4.3.2.4 stack_push()

```
void stack_push (
        struct stack_node * stack,
        struct stack_node * node )
```

Push a node on top of the stack.

Parameters

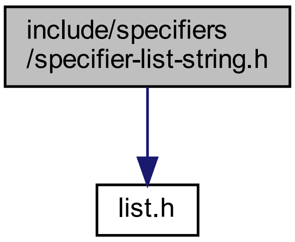
stack	Top node of the stack
node	Node to push

4.4 include/specifiers/specifier-list-string.h File Reference

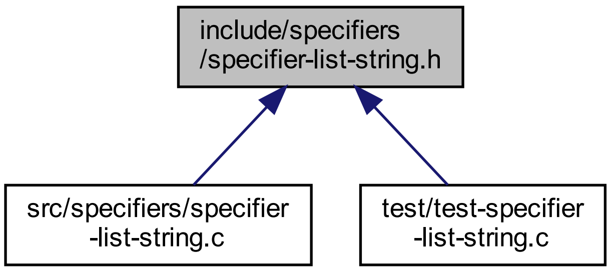
List data structure specifier for string values.

```
#include "list.h"
```

Include dependency graph for specifier-list-string.h:



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



Functions

- struct list_node ∗ list_string_alloc_node (const char ∗str)

  *Allocate a new node with the string **str** value.*
- void list_string_free_node (struct list_node ∗node)

  Free a string node.
- char ∗ list_string_read_node (struct list_node ∗node)

  Reads the string value in a string node.
- int list_string_equal (void ∗left, void ∗right)

4.4.1 Detailed Description

List data structure specifier for string values.

Author

Sébastien Goubeau (`sebastien.goubeau@protonmail.com`)

Containes all the functions to allocate a list node, free it, read a string from it or compare it.

4.4.2 Function Documentation

4.4.2.1 list_string_alloc_node()

```
struct list_node* list_string_alloc_node (
        const char * str )
```

Allocate a new node with the string **str** value.

Parameters

str	String to copy in the node

Returns

The just allocated node

There are two allocation here, one for the list_node structure and one for the str argument.

4.4.2.2 list_string_equal()

```
int list_string_equal (
        void * left,
        void * right )
```

4.4.2.3 list_string_free_node()

```
void list_string_free_node (
        struct list_node * node )
```

Free a string node.

Parameters

node	The node to free

There are two free done here, one for the list_node structure and one for the value (string) of the node.

4.4.2.4 list_string_read_node()

```
char* list_string_read_node (
        struct list_node * node )
```

Reads the string value in a string node.

Parameters

node	Node to read from

Returns

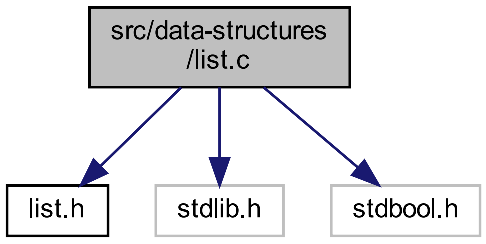
The pointer to the string in the node

The pointer returned is from the node it self, no copy is done here.

4.5 src/data-structures/list.c File Reference

```
#include "list.h"
#include <stdlib.h>
#include <stdbool.h>
```

Include dependency graph for list.c:



Functions

- void list_init (struct list ∗list)

    *Initiate the **list** structure with default values*
    No allocation is done here
- unsigned list_size (struct list ∗list)

    Get the size of the list.
- int list_isEmpty (struct list ∗list)

    Check whether the list is empty.
- void list_hard_append (struct list ∗list, struct list_node ∗node)
- void list_append (struct list ∗list, struct list_node ∗node)

    Append a node to a list.
- void list_prepend (struct list ∗list, struct list_node ∗node)

    Prepend a node to a list.
- struct list_node ∗ list_hard_remove (struct list ∗list, struct list_node ∗node)
- struct list_node ∗ list_popleft (struct list ∗list)

    Pop the left node of a list
    No free is done here
- struct list_node ∗ list_popright (struct list ∗list)

    Pop the right node of a list
    No free is done here
- struct list_node ∗ list_get (struct list ∗list, unsigned index)

    Get a node from a list.
- int list_insert (struct list ∗list, struct list_node ∗node, unsigned index)

    Insert a node at an index of a list.
- struct list_node ∗ list_remove (struct list ∗list, unsigned index)

    Remove a node from a list
    No free is done here

4.5.1 Function Documentation

4.5.1.1 list_append()

```
void list_append (
        struct list * list,
        struct list_node * node )
```

Append a node to a list.

Parameters

list	List to append from
node	Node to append

4.5.1.2 list_get()

```
struct list_node* list_get (
        struct list * list,
        unsigned index )
```

Get a node from a list.

Parameters

list	List from which to get the node from
index	Index of the node to fetch from the left

Returns

Node's pointer

4.5.1.3 list_hard_append()

```
void list_hard_append (
        struct list * list,
        struct list_node * node )
```

4.5.1.4 list_hard_remove()

```
struct list_node* list_hard_remove (
        struct list * list,
        struct list_node * node )
```

4.5.1.5 list_init()

```
void list_init (
        struct list * list )
```

Initiate the **list** structure with default values
No allocation is done here

Parameters

list	List structure

4.5.1.6 list_insert()

```
int list_insert (
        struct list * list,
        struct list_node * node,
        unsigned index )
```

Insert a node at an index of a list.

Parameters

list	List from which to insert to
node	Node to insert
index	Index of the new node in the **list**

Returns

true (1) if the node has been inserted, false (0) otherwise. The function returns false if the index is greater than the list's size.

4.5.1.7 list_isEmpty()

```
int list_isEmpty (
        struct list * list )
```

Check whether the list is empty.

Parameters

list	List of which to check emptiness

Returns

true (1) if the **list** is empty, false (0) otherwise

4.5.1.8 list_popleft()

```
struct list_node* list_popleft (
        struct list * list )
```

Pop the left node of a list
No free is done here

Parameters

list	List from which to pop from

Returns

The node that just been poped

4.5.1.9 list_popright()

```
struct list_node* list_popright (
        struct list * list )
```

Pop the right node of a list
No free is done here

Parameters

list	List from which to pop from

Returns

The node that just been poped

4.5.1.10 list_prepend()

```
void list_prepend (
        struct list * list,
        struct list_node * node )
```

Prepend a node to a list.

Parameters

list	List to prepend from
node	Node to prepend

4.5.1.11 list_remove()

```
struct list_node* list_remove (
        struct list * list,
        unsigned index )
```

Remove a node from a list
No free is done here

Parameters

list	List from which to remove from
index	Index of the node to remove

Returns

The node that just been removed. The function returns NULL if the index is greater or equal to the list's size.

4.5.1.12 list_size() `unsigned list_size ( struct list * list )`

Get the size of the list.

Parameters

list	List of which to get the size from

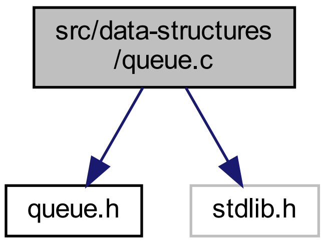
Returns

Size of the list **list**

4.6 src/data-structures/queue.c File Reference

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

Include dependency graph for queue.c:



Functions

- void queue_init (struct queue *q)

  *Initiate the **queue** structure with default values*
  No allocation is done here
- unsigned queue_size (struct queue *q)

  Get the size of the queue.
- int queue_isEmpty (struct queue *q)

  Check whether the queue is empty.
- void queue_enqueue (struct queue *q, struct queue_node *n)

  Enqueue a node to a queue.
- struct queue_node * queue_dequeue (struct queue *q)

  Dequeue a node from a queue
  No free is done here

4.6.1 Function Documentation

4.6.1.1 queue_dequeue()

```
struct queue_node* queue_dequeue (
        struct queue * q )
```

Dequeue a node from a queue
No free is done here

Parameters

q	Queue to dequeue from

Returns

The node that just been dequeued. Returns NULL if the queue is empty.

4.6.1.2 queue_enqueue()

```
void queue_enqueue (
        struct queue * q,
        struct queue_node * n )
```

Enqueue a node to a queue.

Parameters

q	Queue to enqueue from
n	Node to enqueue

4.6.1.3 queue_init()

```
void queue_init (
        struct queue * q )
```

Initiate the **queue** structure with default values
No allocation is done here

Parameters

q	Queue structure

4.6.1.4 queue_isEmpty()

```
int queue_isEmpty (
        struct queue * q )
```

Check whether the queue is empty.

Parameters

q	Queue of which to check emptiness

Returns

true (1) if the **queue** is empty, false (0) otherwise

4.6.1.5 queue_size()

```
unsigned queue_size (
    struct queue * q )
```

Get the size of the queue.

Parameters

q	Queue of which to get the size from

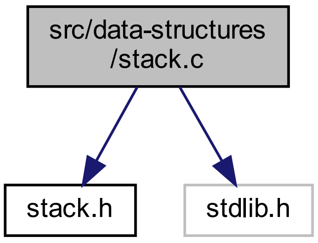
Returns

Size of the queue **queue**

4.7 src/data-structures/stack.c File Reference

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

Include dependency graph for stack.c:



Functions

- void stack_initStack (struct stack_node ∗stack)

  *Initiate the **stack** structure with default vales*
  No allocation is done here

- int stack_isStackEmpty (struct stack_node ∗stack)

  Check whether the stack is empty.
- void stack_push (struct stack_node ∗stack, struct stack_node ∗node)

  Push a node on top of the stack.
- struct stack_node ∗ stack_pop (struct stack_node ∗stack)

  Pop the node on top of the stack
  No free is done here

4.7.1 Function Documentation

4.7.1.1 stack_initStack()

```
void stack_initStack (
        struct stack_node * stack )
```

Initiate the **stack** structure with default vales
No allocation is done here

Parameters

stack	Top node of the stack

The first node of the stack can be considered the stack it self in this implementation.

4.7.1.2 stack_isStackEmpty()

```
int stack_isStackEmpty (
        struct stack_node * stack )
```

Check whether the stack is empty.

Parameters

stack	Top node of the stack

Returns

true (1) if the **stack** is empty, false (0) otherwise

4.7.1.3 stack_pop()

```
struct stack_node* stack_pop (
        struct stack_node * stack )
```

Pop the node on top of the stack
No free is done here

Parameters

stack	Top node of the stack

Returns

The node that just been poped

4.7.1.4 stack_push()

```
void stack_push (
        struct stack_node * stack,
        struct stack_node * node )
```

Push a node on top of the stack.

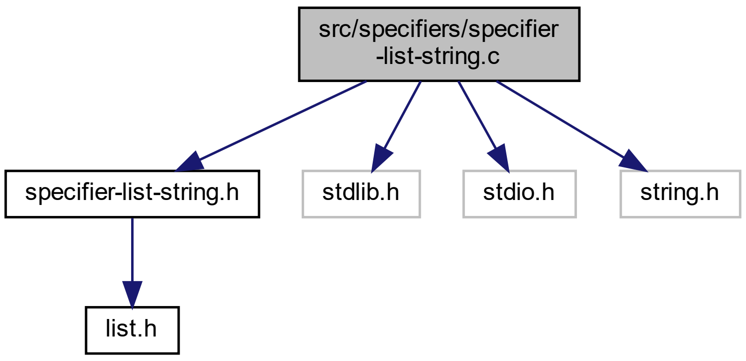
Parameters

stack	Top node of the stack
node	Node to push

4.8 src/specifiers/specifier-list-string.c File Reference

```
#include "specifier-list-string.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
```

Include dependency graph for specifier-list-string.c:



Functions

- struct list_node ∗ list_string_alloc_node (const char ∗str)

    *Allocate a new node with the string **str** value.*
- void list_string_free_node (struct list_node ∗node)

    Free a string node.
- char ∗ list_string_read_node (struct list_node ∗node)

    Reads the string value in a string node.
- int list_string_equal (void ∗left, void ∗right)

4.8.1 Function Documentation

4.8.1.1 list_string_alloc_node()

```
struct list_node* list_string_alloc_node (
        const char * str )
```

Allocate a new node with the string **str** value.

Parameters

str	String to copy in the node

Returns

The just allocated node

There are two allocation here, one for the list_node structure and one for the str argument.

4.8.1.2 list_string_equal()

```
int list_string_equal (
        void * left,
        void * right )
```

4.8.1.3 list_string_free_node()

```
void list_string_free_node (
        struct list_node * node )
```

Free a string node.

Parameters

node	The node to free

There are two free done here, one for the list_node structure and one for the value (string) of the node.

4.8.1.4 list_string_read_node()

```
char* list_string_read_node (
        struct list_node * node )
```

Reads the string value in a string node.

Parameters

node	Node to read from

Returns

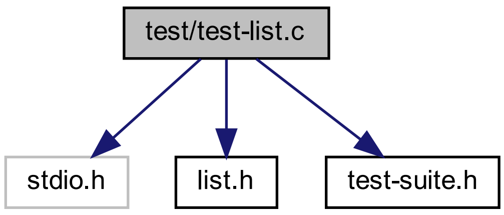
The pointer to the string in the node

The pointer returned is from the node it self, no copy is done here.

4.9 test/test-list.c File Reference

```
#include <stdio.h>
#include "list.h"
#include "test-suite.h"
```

Include dependency graph for test-list.c:



Functions

- struct test_rate test_list ()

4.9.1 Function Documentation

4.9.1.1 test_list()

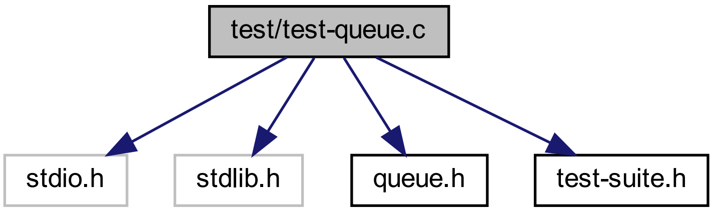
```
struct test_rate test_list (
    void  )
```

4.10 test/test-queue.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"
```

```
#include "test-suite.h"
```

Include dependency graph for test-queue.c:



Functions

- struct test_rate test_queue ()

4.10.1 Function Documentation

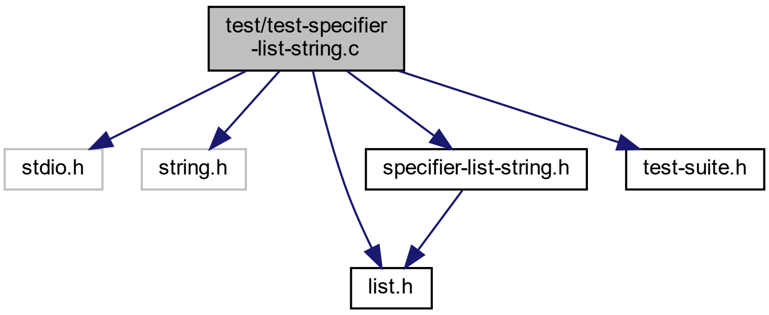
4.10.1.1 test_queue()

```
struct test_rate test_queue (
        void  )
```

4.11 test/test-specifier-list-string.c File Reference

```
#include <stdio.h>
#include <string.h>
#include "list.h"
#include "specifier-list-string.h"
#include "test-suite.h"
```

Include dependency graph for test-specifier-list-string.c:



Functions

- struct test_rate test_list_string ()

4.11.1 Function Documentation

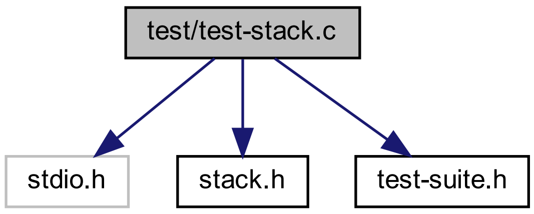
4.11.1.1 test_list_string()

```
struct test_rate test_list_string (
            void  )
```

4.12 test/test-stack.c File Reference

```
#include <stdio.h>
#include "stack.h"
#include "test-suite.h"
```

Include dependency graph for test-stack.c:



Functions

- struct test_rate test_stack ()

4.12.1 Function Documentation

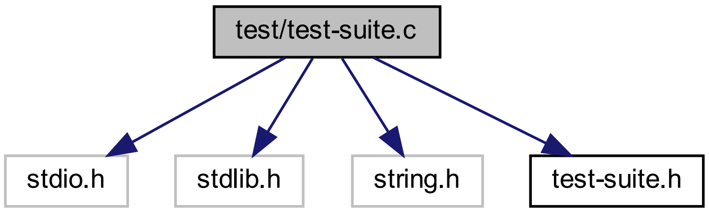
4.12.1.1 test_stack()

```
struct test_rate test_stack (
            void  )
```

4.13 test/test-suite.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "test-suite.h"
```

Include dependency graph for test-suite.c:



Macros

- #define ANSI_COLOR_RED "\x1b[31m"
- #define ANSI_COLOR_GREEN "\x1b[32m"
- #define ANSI_COLOR_RESET "\x1b[0m"

Functions

- struct test_rate test ()
- void test_assert (struct test_rate ∗rate, const char ∗name, int assert)
- void test_print_rate (unsigned tests, unsigned succes)
- int main ()

4.13.1 Macro Definition Documentation

4.13.1.1 ANSI_COLOR_GREEN `#define ANSI_COLOR_GREEN "\x1b[32m"`

4.13.1.2 ANSI_COLOR_RED `#define ANSI_COLOR_RED "\x1b[31m"`

4.13.1.3 ANSI_COLOR_RESET `#define ANSI_COLOR_RESET "\x1b[0m"`

4.13.2 Function Documentation

4.13.2.1 main() `int main ( )`

4.13.2.2 test() `struct test_rate test ( )`

4.13.2.3 test_assert() `void test_assert (`

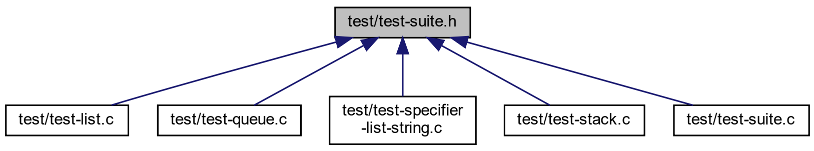
```
struct test_rate * rate,
const char * name,
int assert )
```

4.13.2.4 test_print_rate() `void test_print_rate (`

```
unsigned tests,
unsigned succes )
```

4.14 test/test-suite.h File Reference

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



Data Structures

- struct test_rate
- struct test_section

Functions

- void test_assert (struct test_rate ∗rate, const char ∗name, int assert)
- void test_print_rate (unsigned tests, unsigned succes)
- struct test_rate test_queue (void)
- struct test_rate test_stack (void)
- struct test_rate test_list (void)
- struct test_rate test_list_string (void)

4.14.1 Function Documentation

4.14.1.1 test_assert()

```
void test_assert (
    struct test_rate * rate,
    const char * name,
    int assert )
```

4.14.1.2 test_list()

```
struct test_rate test_list (
    void )
```

4.14.1.3 test_list_string()

```
struct test_rate test_list_string (
    void )
```

4.14.1.4 test_print_rate()

```
void test_print_rate (
    unsigned tests,
    unsigned succes )
```

4.14.1.5 test_queue()

```
struct test_rate test_queue (
    void )
```

4.14.1.6 test_stack()

```
struct test_rate test_stack (
    void )
```

Index