libgenerics

Generated by Doxygen 1.8.13

Contents

1 Class Index **1**

1.1 Class List 1

2 File Index **3**

2.1 File List 3

3 Class Documentation **5**

3.1 graph_t Struct Reference 5

3.1.1 Detailed Description 6

3.1.2 Member Data Documentation 6

3.1.2.1 adj 6

3.1.2.2 E 6

3.1.2.3 label 6

3.1.2.4 member_size 6

3.1.2.5 V 6

3.2 priority_queue_t Struct Reference 7

3.2.1 Member Data Documentation 7

3.2.1.1 compare 7

3.2.1.2 compare_argument 7

3.2.1.3 member_size 7

3.2.1.4 queue 8

3.2.1.5 size 8

3.3 qnode_t Struct Reference 8

3.3.1 Detailed Description 8

3.3.2 Member Data Documentation . . . 8
3.3.2.1 data . . . 9
3.3.2.2 next . . . 9
3.3.2.3 prev . . . 9
3.4 queue_t Struct Reference . . . 9
3.4.1 Detailed Description . . . 10
3.4.2 Member Data Documentation . . . 10
3.4.2.1 head . . . 10
3.4.2.2 member_size . . . 10
3.4.2.3 size . . . 10
3.4.2.4 tail . . . 10
3.5 redblacknode_t Struct Reference . . . 10
3.5.1 Member Data Documentation . . . 11
3.5.1.1 color . . . 11
3.5.1.2 data . . . 11
3.5.1.3 left . . . 11
3.5.1.4 parent . . . 11
3.5.1.5 right . . . 11
3.6 redblacktree_t Struct Reference . . . 12
3.6.1 Member Data Documentation . . . 12
3.6.1.1 compare . . . 12
3.6.1.2 compare_argument . . . 12
3.6.1.3 member_size . . . 13
3.6.1.4 root . . . 13
3.6.1.5 size . . . 13
3.7 snode_t Struct Reference . . . 13
3.7.1 Detailed Description . . . 13
3.7.2 Member Data Documentation . . . 14
3.7.2.1 data . . . 14
3.7.2.2 next . . . 14

3.7.2.3 prev . . . 14
3.8 stack_t Struct Reference . . . 14
3.8.1 Detailed Description . . . 15
3.8.2 Member Data Documentation . . . 15
3.8.2.1 head . . . 15
3.8.2.2 member_size . . . 15
3.8.2.3 size . . . 15
3.9 tnode_t Struct Reference . . . 15
3.9.1 Detailed Description . . . 16
3.9.2 Member Data Documentation . . . 16
3.9.2.1 children . . . 16
3.9.2.2 value . . . 16
3.10 trie_t Struct Reference . . . 16
3.10.1 Detailed Description . . . 17
3.10.2 Member Data Documentation . . . 17
3.10.2.1 member_size . . . 17
3.10.2.2 root . . . 17
3.10.2.3 size . . . 17
3.11 vector_t Struct Reference . . . 17
3.11.1 Member Data Documentation . . . 18
3.11.1.1 buffer_size . . . 18
3.11.1.2 data . . . 18
3.11.1.3 member_size . . . 18
3.11.1.4 size . . . 18

4 File Documentation **19**

4.1 include/gerror.h File Reference . . . 19

4.1.1 Typedef Documentation . . . 20

4.1.1.1 gerror_t . . . 20

4.1.2 Enumeration Type Documentation . . . 20

4.1.2.1 gerror_t . . . 20

4.1.3 Function Documentation . . . 21

4.1.3.1 gerror_to_str() . . . 21

4.2 include/graph.h File Reference . . . 21

4.2.1 Typedef Documentation . . . 22

4.2.1.1 graph_t . . . 22

4.2.2 Function Documentation . . . 22

4.2.2.1 graph_add_edge() . . . 22

4.2.2.2 graph_create() . . . 23

4.2.2.3 graph_destroy() . . . 23

4.2.2.4 graph_get_label_at() . . . 23

4.2.2.5 graph_set_label_at() . . . 25

4.3 include/priority_queue.h File Reference . . . 25

4.3.1 Typedef Documentation . . . 27

4.3.1.1 pqueue_compare_function . . . 27

4.3.1.2 pqueue_t . . . 27

4.3.1.3 priority_queue_t . . . 27

4.3.2 Enumeration Type Documentation . . . 27

4.3.2.1 queue_priority_t . . . 27

4.3.3 Function Documentation . . . 28

4.3.3.1 pqueue_add() . . . 28

4.3.3.2 pqueue_create() . . . 28

4.3.3.3 pqueue_destroy() . . . 28

4.3.3.4 pqueue_extract() . . . 29

4.3.3.5 pqueue_max_priority() . . . 29

4.3.3.6 pqueue_set_compare_function() . . . 30
4.4 include/queue.h File Reference . . . 30
4.4.1 Typedef Documentation . . . 31
4.4.1.1 qnode_t . . . 31
4.4.1.2 queue_t . . . 32
4.4.2 Function Documentation . . . 32
4.4.2.1 queue_create() . . . 32
4.4.2.2 queue_dequeue() . . . 32
4.4.2.3 queue_destroy() . . . 33
4.4.2.4 queue_enqueue() . . . 33
4.4.2.5 queue_remove() . . . 33
4.5 include/red_black_tree.h File Reference . . . 34
4.5.1 Typedef Documentation . . . 35
4.5.1.1 rbnode_t . . . 35
4.5.1.2 rbtree_compare_function . . . 35
4.5.1.3 rbtree_t . . . 35
4.5.1.4 redblacknode_t . . . 36
4.5.1.5 redblacktree_t . . . 36
4.5.2 Enumeration Type Documentation . . . 36
4.5.2.1 rbcolor_t . . . 36
4.5.2.2 rbcomp_t . . . 36
4.5.3 Function Documentation . . . 36
4.5.3.1 rbtree_add() . . . 36
4.5.3.2 rbtree_create() . . . 37
4.5.3.3 rbtree_destroy() . . . 37
4.5.3.4 rbtree_find_node() . . . 38
4.5.3.5 rbtree_max_node() . . . 38
4.5.3.6 rbtree_max_value() . . . 38
4.5.3.7 rbtree_min_node() . . . 39
4.5.3.8 rbtree_min_value() . . . 39

4.5.3.9 rbtree_remove_item() . . . 40
4.5.3.10 rbtree_remove_node() . . . 40
4.5.3.11 rbtree_set_compare_function() . . . 41
4.6 include/stack.h File Reference . . . 41
4.6.1 Typedef Documentation . . . 42
4.6.1.1 snode_t . . . 42
4.6.1.2 stack_t . . . 42
4.6.2 Function Documentation . . . 43
4.6.2.1 stack_create() . . . 43
4.6.2.2 stack_destroy() . . . 43
4.6.2.3 stack_pop() . . . 43
4.6.2.4 stack_push() . . . 44
4.7 include/trie.h File Reference . . . 44
4.7.1 Macro Definition Documentation . . . 46
4.7.1.1 NBYTE . . . 46
4.7.2 Typedef Documentation . . . 46
4.7.2.1 tnode_t . . . 46
4.7.2.2 trie_t . . . 46
4.7.3 Function Documentation . . . 46
4.7.3.1 trie_add_element() . . . 46
4.7.3.2 trie_create() . . . 47
4.7.3.3 trie_destroy() . . . 47
4.7.3.4 trie_get_element() . . . 47
4.7.3.5 trie_get_node_or_allocate() . . . 48
4.7.3.6 trie_remove_element() . . . 48
4.7.3.7 trie_set_element() . . . 48
4.8 include/vector.h File Reference . . . 49
4.8.1 Typedef Documentation . . . 50
4.8.1.1 vector_t . . . 50
4.8.2 Function Documentation . . . 51

4.8.2.1 vector_add() . . . 51
4.8.2.2 vector_at() . . . 51
4.8.2.3 vector_create() . . . 51
4.8.2.4 vector_destroy() . . . 52
4.8.2.5 vector_get_min_buf_siz() . . . 52
4.8.2.6 vector_ptr_at() . . . 53
4.8.2.7 vector_resize_buffer() . . . 53
4.8.2.8 vector_set_elem_at() . . . 53
4.8.2.9 vector_set_min_buf_siz() . . . 54
4.9 src/gerror.c File Reference . . . 54
4.9.1 Function Documentation . . . 55
4.9.1.1 gerror_to_str() . . . 55
4.9.2 Variable Documentation . . . 55
4.9.2.1 gerror_to_string . . . 55
4.10 src/graph.c File Reference . . . 55
4.10.1 Function Documentation . . . 56
4.10.1.1 graph_add_edge() . . . 56
4.10.1.2 graph_create() . . . 57
4.10.1.3 graph_destroy() . . . 57
4.10.1.4 graph_get_label_at() . . . 58
4.10.1.5 graph_set_label_at() . . . 58
4.11 src/priority_queue.c File Reference . . . 58
4.11.1 Macro Definition Documentation . . . 59
4.11.1.1 LEFT . . . 60
4.11.1.2 PARENT . . . 60
4.11.1.3 RIGHT . . . 60
4.11.2 Function Documentation . . . 60
4.11.2.1 max_heapify() . . . 60
4.11.2.2 nswap() . . . 60
4.11.2.3 pqueue_add() . . . 60

4.11.2.4 pqueue_create() . . . 61
4.11.2.5 pqueue_default_compare_function() . . . 61
4.11.2.6 pqueue_destroy() . . . 61
4.11.2.7 pqueue_extract() . . . 62
4.11.2.8 pqueue_max_priority() . . . 62
4.11.2.9 pqueue_set_compare_function() . . . 63
4.12 src/queue.c File Reference . . . 63
4.12.1 Function Documentation . . . 64
4.12.1.1 queue_create() . . . 64
4.12.1.2 queue_dequeue() . . . 64
4.12.1.3 queue_destroy() . . . 65
4.12.1.4 queue_enqueue() . . . 65
4.12.1.5 queue_remove() . . . 65
4.13 src/red_black_tree.c File Reference . . . 66
4.13.1 Enumeration Type Documentation . . . 67
4.13.1.1 rbc_t . . . 67
4.13.2 Function Documentation . . . 67
4.13.2.1 create_node() . . . 67
4.13.2.2 fix_insert_case() . . . 68
4.13.2.3 left_rotate() . . . 68
4.13.2.4 rbnode_destroy() . . . 68
4.13.2.5 rbnode_is_black() . . . 68
4.13.2.6 rbnode_is_red() . . . 68
4.13.2.7 rbtree_add() . . . 68
4.13.2.8 rbtree_create() . . . 69
4.13.2.9 rbtree_create_double_black() . . . 69
4.13.2.10 rbtree_default_compare_function() . . . 69
4.13.2.11 rbtree_delete_fixup() . . . 70
4.13.2.12 rbtree_destroy() . . . 70
4.13.2.13 rbtree_find_minimal_node() . . . 70

4.13.2.14 rbtree_find_node() . . . 70
4.13.2.15 rbtree_identify_case() . . . 71
4.13.2.16 rbtree_insert_fixup() . . . 71
4.13.2.17 rbtree_max_node() . . . 71
4.13.2.18 rbtree_max_value() . . . 71
4.13.2.19 rbtree_min_node() . . . 73
4.13.2.20 rbtree_min_value() . . . 73
4.13.2.21 rbtree_remove_double_black() . . . 74
4.13.2.22 rbtree_remove_item() . . . 74
4.13.2.23 rbtree_remove_node() . . . 74
4.13.2.24 rbtree_set_compare_function() . . . 75
4.13.2.25 rbtree_transplant() . . . 75
4.13.2.26 right_rotate() . . . 75
4.14 src/stack.c File Reference . . . 76
4.14.1 Function Documentation . . . 76
4.14.1.1 stack_create() . . . 76
4.14.1.2 stack_destroy() . . . 77
4.14.1.3 stack_pop() . . . 77
4.14.1.4 stack_push() . . . 77
4.15 src/trie.c File Reference . . . 78
4.15.1 Function Documentation . . . 79
4.15.1.1 node_at() . . . 79
4.15.1.2 trie_add_element() . . . 79
4.15.1.3 trie_create() . . . 79
4.15.1.4 trie_destroy() . . . 80
4.15.1.5 trie_destroy_tnode() . . . 80
4.15.1.6 trie_get_element() . . . 80
4.15.1.7 trie_get_node_or_allocate() . . . 80
4.15.1.8 trie_remove_element() . . . 81
4.15.1.9 trie_set_element() . . . 81

4.16 src/vector.c File Reference 81
4.16.1 Macro Definition Documentation 82
4.16.1.1 VECTOR_MIN_SIZ 83
4.16.2 Function Documentation 83
4.16.2.1 vector_add() 83
4.16.2.2 vector_at() 83
4.16.2.3 vector_create() 84
4.16.2.4 vector_destroy() 84
4.16.2.5 vector_get_min_buf_siz() 84
4.16.2.6 vector_ptr_at() 85
4.16.2.7 vector_resize_buffer() 85
4.16.2.8 vector_set_elem_at() 85
4.16.2.9 vector_set_min_buf_siz() 86
4.16.3 Variable Documentation 86
4.16.3.1 vector_min_siz 86

Index **87**

Chapter 1

Class Index

1.1 Class List

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

graph_t 5
priority_queue_t 7
qnode_t 8
queue_t 9
redblacknode_t 10
redblacktree_t 12
snode_t 13
stack_t 14
tnode_t 15
trie_t 16
vector_t 17

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

include/gerror.h . . . 19
include/graph.h . . . 21
include/priority_queue.h . . . 25
include/queue.h . . . 30
include/red_black_tree.h . . . 34
include/stack.h . . . 41
include/trie.h . . . 44
include/vector.h . . . 49
src/gerror.c . . . 54
src/graph.c . . . 55
src/priority_queue.c . . . 58
src/queue.c . . . 63
src/red_black_tree.c . . . 66
src/stack.c . . . 76
src/trie.c . . . 78
src/vector.c . . . 81

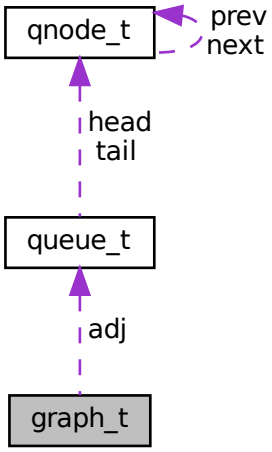
Chapter 3

Class Documentation

3.1 graph_t Struct Reference

`#include <graph.h>`

Collaboration diagram for graph_t:



Public Attributes

- size_t V
- size_t E
- size_t member_size
- struct queue_t * adj
- void * label

3.1.1 Detailed Description

Graph structure and elements.

3.1.2 Member Data Documentation

3.1.2.1 adj

```
struct queue_t* graph_t::adj
```

3.1.2.2 E

```
size_t graph_t::E
```

3.1.2.3 label

```
void* graph_t::label
```

3.1.2.4 member_size

```
size_t graph_t::member_size
```

3.1.2.5 V

```
size_t graph_t::V
```

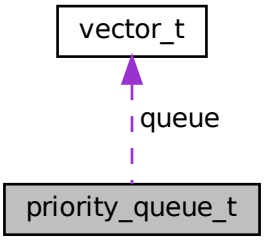
The documentation for this struct was generated from the following file:

- include/graph.h

3.2 priority_queue_t Struct Reference

`#include <priority_queue.h>`

Collaboration diagram for priority_queue_t:



Public Attributes

- size_t size
- size_t member_size
- pqueue_compare_function compare
- void * compare_argument
- struct vector_t queue

3.2.1 Member Data Documentation

3.2.1.1 compare

`pqueue_compare_function priority_queue_t::compare`

3.2.1.2 compare_argument

`void* priority_queue_t::compare_argument`

3.2.1.3 member_size

`size_t priority_queue_t::member_size`

3.2.1.4 queue

```
struct vector_t priority_queue_t::queue
```

3.2.1.5 size

```
size_t priority_queue_t::size
```

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

- include/priority_queue.h

3.3 qnode_t Struct Reference

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

Collaboration diagram for qnode_t:

Public Attributes

- struct qnode_t * next
- struct qnode_t * prev
- void * data

3.3.1 Detailed Description

queue node.

3.3.2 Member Data Documentation

3.3.2.1 data

```
void* qnode_t::data
```

3.3.2.2 next

```
struct qnode_t* qnode_t::next
```

3.3.2.3 prev

```
struct qnode_t* qnode_t::prev
```

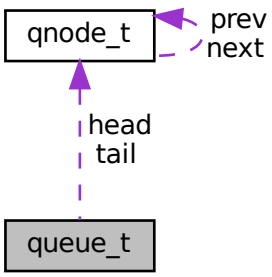
The documentation for this struct was generated from the following file:

- include/queue.h

3.4 queue_t Struct Reference

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

Collaboration diagram for queue_t:



Public Attributes

- size_t size
- size_t member_size
- struct qnode_t * head
- struct qnode_t * tail

3.4.1 Detailed Description

Represents a queue structure.

3.4.2 Member Data Documentation

3.4.2.1 head

```
struct qnode_t* queue_t::head
```

3.4.2.2 member_size

```
size_t queue_t::member_size
```

3.4.2.3 size

```
size_t queue_t::size
```

3.4.2.4 tail

```
struct qnode_t* queue_t::tail
```

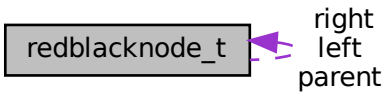
The documentation for this struct was generated from the following file:

- include/queue.h

3.5 redblacknode_t Struct Reference

```
#include <red_black_tree.h>
```

Collaboration diagram for redblacknode_t:



Public Attributes

- struct redblacknode_t * left
- struct redblacknode_t * right
- struct redblacknode_t * parent
- rbcolor_t color
- void * data

3.5.1 Member Data Documentation

3.5.1.1 color

```
rbcolor_t redblacknode_t::color
```

3.5.1.2 data

```
void* redblacknode_t::data
```

3.5.1.3 left

```
struct redblacknode_t* redblacknode_t::left
```

3.5.1.4 parent

```
struct redblacknode_t* redblacknode_t::parent
```

3.5.1.5 right

```
struct redblacknode_t* redblacknode_t::right
```

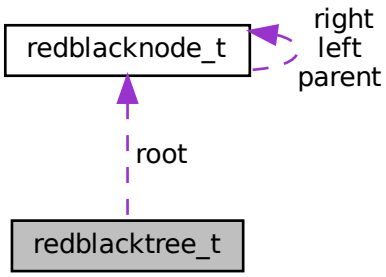
The documentation for this struct was generated from the following file:

- include/red_black_tree.h

3.6 redblacktree_t Struct Reference

`#include <red_black_tree.h>`

Collaboration diagram for redblacktree_t:



Public Attributes

- size_t size
- size_t member_size
- rbtree_compare_function compare
- void * compare_argument
- struct redblacknode_t * root

3.6.1 Member Data Documentation

3.6.1.1 compare

`rbtree_compare_function redblacktree_t::compare`

3.6.1.2 compare_argument

`void* redblacktree_t::compare_argument`

3.6.1.3 member_size

```
size_t redblacktree_t::member_size
```

3.6.1.4 root

```
struct redblacknode_t* redblacktree_t::root
```

3.6.1.5 size

```
size_t redblacktree_t::size
```

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

- include/red_black_tree.h

3.7 snode_t Struct Reference

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

Collaboration diagram for snode_t:

Public Attributes

- struct snode_t * next
- struct snode_t * prev
- void * data

3.7.1 Detailed Description

node of a stack

3.7.2 Member Data Documentation

3.7.2.1 data

```
void* snode_t::data
```

3.7.2.2 next

```
struct snode_t* snode_t::next
```

3.7.2.3 prev

```
struct snode_t* snode_t::prev
```

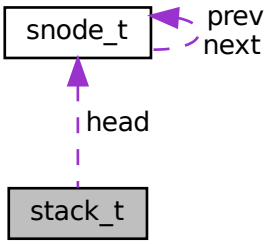
The documentation for this struct was generated from the following file:

- include/stack.h

3.8 stack_t Struct Reference

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

Collaboration diagram for stack_t:



Public Attributes

- size_t size
- size_t member_size
- struct snode_t * head

3.8.1 Detailed Description

represents the stack structure.

3.8.2 Member Data Documentation

3.8.2.1 head

```
struct snode_t* stack_t::head
```

3.8.2.2 member_size

```
size_t stack_t::member_size
```

3.8.2.3 size

```
size_t stack_t::size
```

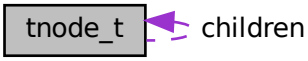
The documentation for this struct was generated from the following file:

- include/stack.h

3.9 tnode_t Struct Reference

```
#include <trie.h>
```

Collaboration diagram for tnode_t:



Public Attributes

- void * value
- struct tnode_t * children [NBYTE]

3.9.1 Detailed Description

node of a trie_t element.

3.9.2 Member Data Documentation

3.9.2.1 children

```
struct tnode_t* tnode_t::children[NBYTE]
```

3.9.2.2 value

```
void* tnode_t::value
```

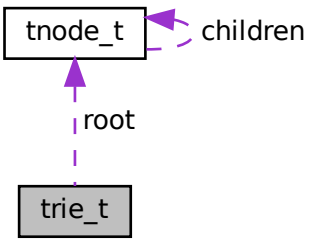
The documentation for this struct was generated from the following file:

- include/trie.h

3.10 trie_t Struct Reference

```
#include <trie.h>
```

Collaboration diagram for trie_t:



Public Attributes

- size_t size
- size_t member_size
- struct tnode_t root

3.10.1 Detailed Description

Represents the trie structure.

3.10.2 Member Data Documentation

3.10.2.1 member_size

```
size_t trie_t::member_size
```

3.10.2.2 root

```
struct tnode_t trie_t::root
```

3.10.2.3 size

```
size_t trie_t::size
```

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

- include/trie.h

3.11 vector_t Struct Reference

```
#include <vector.h>
```

Public Attributes

- void * data
- size_t size
- size_t buffer_size
- size_t member_size

3.11.1 Member Data Documentation

3.11.1.1 buffer_size

```
size_t vector_t::buffer_size
```

3.11.1.2 data

```
void* vector_t::data
```

3.11.1.3 member_size

```
size_t vector_t::member_size
```

3.11.1.4 size

```
size_t vector_t::size
```

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

- include/vector.h

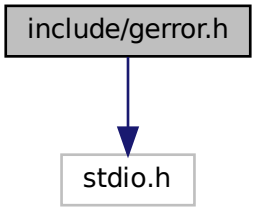
Chapter 4

File Documentation

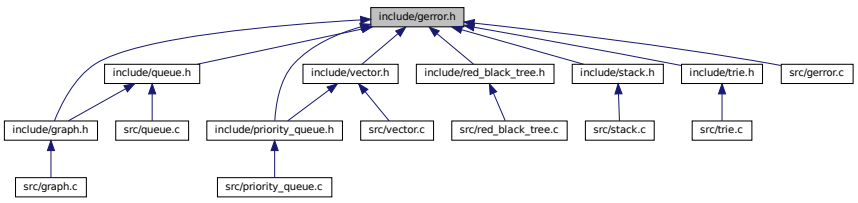
4.1 include/gerror.h File Reference

```
#include <stdio.h>
```

Include dependency graph for gerror.h:



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



Typedefs

- typedef enum gerror_t gerror_t

Enumerations

- enum gerror_t {
  GERROR_OK, GERROR_NULL_STRUCTURE, GERROR_NULL_HEAD, GERROR_NULL_NODE,
  GERROR_NULL_RETURN_POINTER, GERROR_EMPTY_STRUCTURE, GERROR_TRY_REMOVE_E↩
  MPTY_STRUCTURE, GERROR_TRY_ADD_EDGE_NO_VERTEX,
  GERROR_ACCESS_OUT_OF_BOUND, GERROR_NULL_ELEMENT_POINTER, GERROR_REMOVE_↩
  ELEMENT_NOT_FOUNDED, GERROR_ELEMENT_NOT_FOUNDED,
  GERROR_N_ERROR }

Functions

- char * gerror_to_str (gerror_t g)

4.1.1 Typedef Documentation

4.1.1.1 gerror_t

```
typedef enum gerror_t gerror_t
```

4.1.2 Enumeration Type Documentation

4.1.2.1 gerror_t

```
enum gerror_t
```

Enumerator

GERROR_OK	
GERROR_NULL_STRUCTURE	
GERROR_NULL_HEAD	
GERROR_NULL_NODE	
GERROR_NULL_RETURN_POINTER	
GERROR_EMPTY_STRUCTURE	
GERROR_TRY_REMOVE_EMPTY_STRUCTURE	
GERROR_TRY_ADD_EDGE_NO_VERTEX	
GERROR_ACCESS_OUT_OF_BOUND	
GERROR_NULL_ELEMENT_POINTER	
GERROR_REMOVE_ELEMENT_NOT_FOUNDED	
GERROR_ELEMENT_NOT_FOUNDED	
GERROR_N_ERROR	

4.1.3 Function Documentation

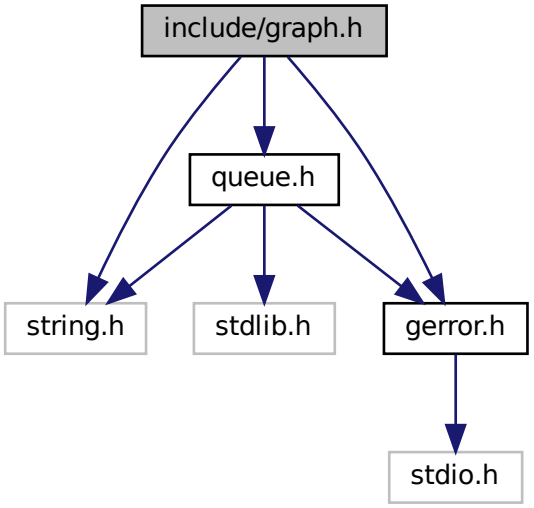
4.1.3.1 gerror_to_str()

```
char* gerror_to_str (
            gerror_t g )
```

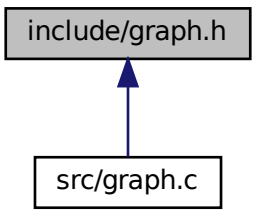
4.2 include/graph.h File Reference

```
#include <string.h>
#include "gerror.h"
#include "queue.h"
```

Include dependency graph for graph.h:



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



Classes

- struct graph_t

Typedefs

- typedef struct graph_t graph_t

Functions

- gerror_t graph_create (graph_t *g, size_t size, size_t member_size)
- gerror_t graph_add_edge (graph_t *g, size_t from, size_t to)
- gerror_t graph_get_label_at (graph_t *g, size_t index, void *label)
- gerror_t graph_set_label_at (graph_t *g, size_t index, void *label)
- gerror_t graph_destroy (graph_t *g)

4.2.1 Typedef Documentation

4.2.1.1 graph_t

```
typedef struct graph_t graph_t
```

Graph structure and elements.

4.2.2 Function Documentation

4.2.2.1 graph_add_edge()

```
gerror_t graph_add_edge (
            graph_t * g,
            size_t from,
            size_t to )
```

Adds an edge on the graph `g` from the vertex `from` to the vertex `to`. Where `from` and `to` are indexes of these vertex.

Parameters

g	pointer to a graph structure;
from	index of the first vertex;
to	index of the incident vertex.

Returns

GERROR_OK in case of success operation; GERROR_TRY_ADD_EDGE_NO_VERTEX in case that `from` or `to` not exists in the graph

4.2.2.2 graph_create()

```
gerror_t graph_create (
            graph_t * g,
            size_t size,
            size_t member_size )
```

Creates a graph and populates the previous allocated structure pointed by `g`;

Parameters

g	pointer to a graph structure;
member_size	size of the elements that will be indexed by `g`

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `g` is a NULL

4.2.2.3 graph_destroy()

```
gerror_t graph_destroy (
            graph_t * g )
```

Deallocates the structures in `g`. This function WILL NOT deallocate the pointer `g`.

Parameters

g	pointer to a graph structure;

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `g` is a NULL

4.2.2.4 graph_get_label_at()

```
gerror_t graph_get_label_at (
            graph_t * g,
```

```
size_t index,
void * label )
```

Gets the label of the vertex in the `index` position of the graph `g`.

Parameters

g	pointer to a graph structure;
index	index of the vertex;
label	pointer to the memory allocated that will be write with the label in `index`

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `g` is a NULL

4.2.2.5 graph_set_label_at()

```
gerror_t graph_set_label_at (
            graph_t * g,
            size_t index,
            void * label )
```

Sets the label at the `index` to `label`.

Parameters

g	pointer to a graph structure;
index	index of the vertex;
label	the new label of the vertex positioned in `index`

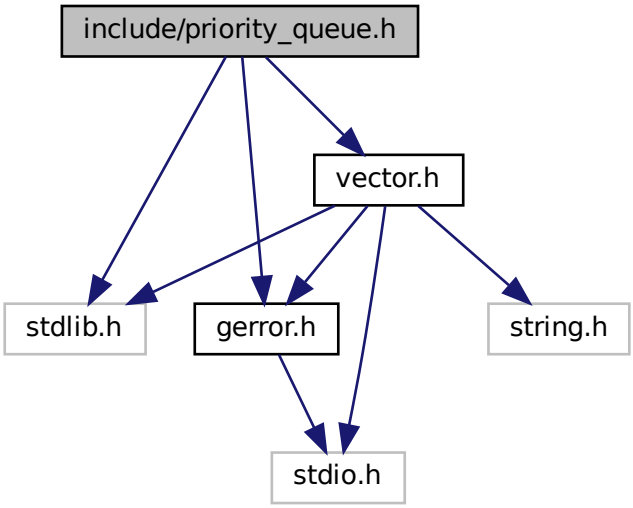
Returns

GERROR_OK in case of success operation; GERROR_ACCESS_OUT_OF_BOUND in case that `index` is out of bound

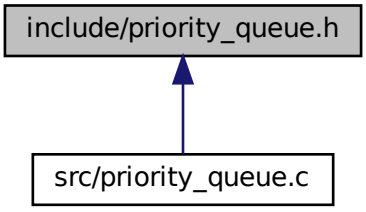
4.3 include/priority_queue.h File Reference

```
#include <stdlib.h>
#include "gerror.h"
#include "vector.h"
```

Include dependency graph for priority_queue.h:



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



Classes

- struct priority_queue_t

Typedefs

- typedef int(∗ pqueue_compare_function) (void ∗a, void ∗b, void ∗arg)
- typedef struct priority_queue_t priority_queue_t
- typedef struct priority_queue_t pqueue_t

Enumerations

- enum queue_priority_t { G_PQUEUE_FIRST_PRIORITY = -1, G_PQUEUE_EQUAL_PRIORITY, G_PQUEUE_SECOND_PRIORITY }

Functions

- gerror_t pqueue_create (pqueue_t *p, size_t member_size)
- gerror_t pqueue_destroy (pqueue_t *p)
- gerror_t pqueue_set_compare_function (pqueue_t *p, pqueue_compare_function function, void *argument)
- gerror_t pqueue_add (pqueue_t *p, void *e)
- gerror_t pqueue_max_priority (pqueue_t *p, void *e)
- gerror_t pqueue_extract (pqueue_t *p, void *e)

4.3.1 Typedef Documentation

4.3.1.1 pqueue_compare_function

```
typedef int(* pqueue_compare_function) (void *a, void *b, void *arg)
```

4.3.1.2 pqueue_t

```
typedef struct priority_queue_t pqueue_t
```

4.3.1.3 priority_queue_t

```
typedef struct priority_queue_t priority_queue_t
```

4.3.2 Enumeration Type Documentation

4.3.2.1 queue_priority_t

```
enum queue_priority_t
```

Enumerator

G_PQUEUE_FIRST_PRIORITY	
G_PQUEUE_EQUAL_PRIORITY	
G_PQUEUE_SECOND_PRIORITY	

4.3.3 Function Documentation

4.3.3.1 pqueue_add()

```
gerror_t pqueue_add (
            pqueue_t * p,
            void * e )
```

Adds an element in the queue and max heap the queue. TODO: A more datailed description of pqueue_add.

Parameters

p	previous allocated pqueue_t struct
e	the element to be added

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `t` is a NULL

4.3.3.2 pqueue_create()

```
gerror_t pqueue_create (
            pqueue_t * p,
            size_t member_size )
```

Populates the `p` structure and inicialize it. A priority queue needs a pqueue_compare_function. The default function will only work for char, int and long. If you need a double or float you need to implement the compare function and set with the function `pqueue_set_compare_function`

Parameters

p	previous allocated pqueue_t struct
member_size	size in bytes of the indexed elements

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `p` is a NULL

4.3.3.3 pqueue_destroy()

```
gerror_t pqueue_destroy (
            pqueue_t * p )
```

Destroy (i.e. deallocates) the `p` structure fields. TODO: A more detailed description of pqueue_destroy.

Parameters

p	previous allocated pqueue_t struct

Returns

TODO

4.3.3.4 pqueue_extract()

```
gerror_t pqueue_extract (
            pqueue_t * p,
            void * e )
```

Extracts the highest priority element in the queue and writes in `e` pointer.

Parameters

p	previous allocated pqueue_t struct
e	pointer to previous allocated variable

Returns

GERROR_OK in case of success operation; GERROR_ACESS_OUT_OF_BOUND in case the queue is empty GERROR_NULL_STRUCURE in case `t` is a NULL

4.3.3.5 pqueue_max_priority()

```
gerror_t pqueue_max_priority (
            pqueue_t * p,
            void * e )
```

Returns and does not remove the highest priority of the queue. TODO: A more datailed description of pqueue_↩ max_priority.

Parameters

p	previous allocated pqueue_t struct
e	pointer to previous allocated variable with `member_size` size that will receive a copy of the highest priority element of the queue.

Returns

GERROR_OK in case of success operation; GERROR_ACESS_OUT_OF_BOUND in case the queue is empty GERROR_NULL_STRUCURE in case `t` is a NULL

4.3.3.6 pqueue_set_compare_function()

```
gerror_t pqueue_set_compare_function (
            pqueue_t * p,
            pqueue_compare_function function,
            void * argument )
```

Change the default comparison function of the priority queue `p` by `function` with the argument `argument`.

Parameters

p	previous allocated pqueue_t struct
function	comparison function callback that has the following prototype: int compare(void* a, void* b) the a and b are the arguments returns -1 if `a` has priority BIG than `b` returns 0 if `a` has priority EQUAL than `b` return 1 if `a` has priority LESS than `b`
argument	pointer to the argument to the comparison function

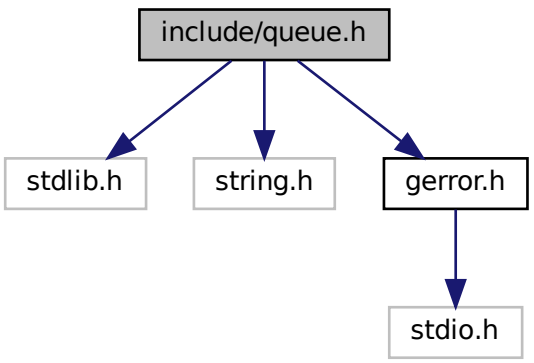
Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `t` is a NULL

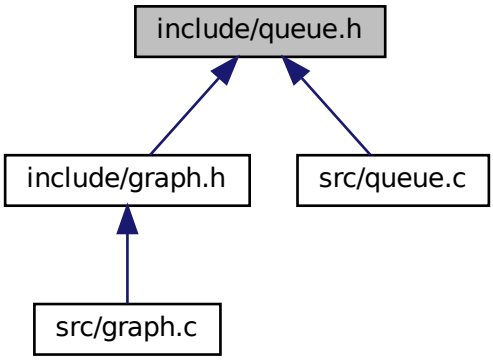
4.4 include/queue.h File Reference

```
#include <stdlib.h>
#include <string.h>
#include "gerror.h"
```

Include dependency graph for queue.h:



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



Classes

- struct qnode_t
- struct queue_t

Typedefs

- typedef struct qnode_t qnode_t
- typedef struct queue_t queue_t

Functions

- gerror_t queue_create (struct queue_t ∗q, size_t member_size)
- gerror_t queue_enqueue (struct queue_t ∗q, void ∗e)
- gerror_t queue_dequeue (struct queue_t ∗q, void ∗e)
- gerror_t queue_destroy (struct queue_t ∗q)
- gerror_t queue_remove (struct queue_t ∗q, struct qnode_t ∗node, void ∗e)

4.4.1 Typedef Documentation

4.4.1.1 qnode_t

```
typedef struct qnode_t qnode_t
```

queue node.

4.4.1.2 queue_t

```
typedef struct queue_t queue_t
```

Represents a queue structure.

4.4.2 Function Documentation

4.4.2.1 queue_create()

```
gerror_t queue_create (
            struct queue_t * q,
            size_t member_size )
```

Creates a queue and populates the previous allocated structure pointed by `q`;

Parameters

q	pointer to a queue structure;
member_size	size of the elements that will be indexed by `q`

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `q` is a NULL pointer

4.4.2.2 queue_dequeue()

```
gerror_t queue_dequeue (
            struct queue_t * q,
            void * e )
```

Dequeues the first element of the queue `q`

Parameters

q	pointer to a queue structure;
e	pointer to the previous allocated element memory that will be write with de dequeued element.

Returns

GERROR_OK in case of success operation; GERROR_NULL_HEAD in case that the head `q->head` is a null pointer. GERROR_NULL_STRUCURE in case `q` is a NULL pointer GERROR_TRY_REMOVE_EMPT↩ Y_STRUCTURE in case that `q` has no element.

4.4.2.3 queue_destroy()

```
gerror_t queue_destroy (
            struct queue_t * q )
```

Deallocate the nodes of the queue q. This function WILL NOT deallocate the pointer q.

Parameters

q	pointer to a queue structure;

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `q` is a NULL pointer

4.4.2.4 queue_enqueue()

```
gerror_t queue_enqueue (
            struct queue_t * q,
            void * e )
```

Enqueues the element pointed by `e` in the queue `q`.

Parameters

q	pointer to a queue structure;
e	pointer to the element that will be indexed by q.

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `q` is a NULL pointer

4.4.2.5 queue_remove()

```
gerror_t queue_remove (
            struct queue_t * q,
            struct qnode_t * node,
            void * e )
```

Removes the element `node` of the queue `q`.

Parameters

q	pointer to a queue structure;
node	element to be removed from the queue
e	pointer to the memory that will be write with the removed element

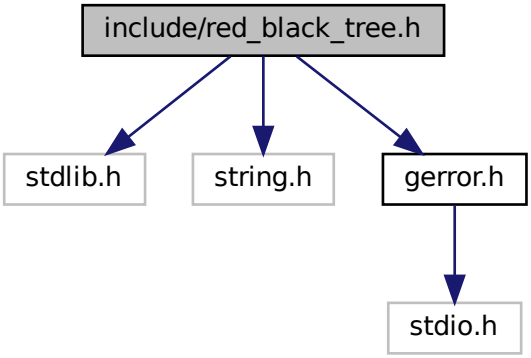
Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `q` is a NULL pointer GERROR_NULL_NODE in case `node` is NULL; GERROR_TRY_REMOVE_EMPTY_STRUCTURE in case that `q` has no element.

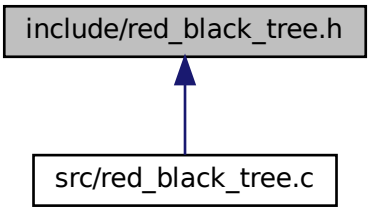
4.5 include/red_black_tree.h File Reference

```
#include <stdlib.h>
#include <string.h>
#include "gerror.h"
```

Include dependency graph for red_black_tree.h:



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



Classes

- struct redblacknode_t
- struct redblacktree_t

Typedefs

- typedef int(∗ rbtree_compare_function) (void ∗a, void ∗b, void ∗arg)
- typedef struct redblacknode_t redblacknode_t
- typedef struct redblacktree_t redblacktree_t
- typedef struct redblacktree_t rbtree_t
- typedef struct redblacknode_t rbnode_t

Enumerations

- enum rbcomp_t { G_RB_FIRST_IS_SMALLER = -1, G_RB_EQUAL, G_RB_FIRST_IS_GREATER }
- enum rbcolor_t { G_RB_RED, G_RB_BLACK, G_RB_DOUBLE_BLACK }

Functions

- gerror_t rbtree_create (rbtree_t ∗rbt, size_t member_size)
- gerror_t rbtree_destroy (rbtree_t ∗rbt)
- gerror_t rbtree_set_compare_function (rbtree_t ∗rbt, rbtree_compare_function function, void ∗argument)
- gerror_t rbtree_add (rbtree_t ∗rbt, void ∗elem)
- gerror_t rbtree_remove_item (rbtree_t ∗rbt, void ∗elem)
- gerror_t rbtree_remove_node (rbtree_t ∗rbt, rbnode_t ∗node)
- gerror_t rbtree_min_node (rbtree_t ∗rbt, rbnode_t ∗∗node)
- gerror_t rbtree_min_value (rbtree_t ∗rbt, void ∗elem)
- gerror_t rbtree_max_node (rbtree_t ∗rbt, rbnode_t ∗∗node)
- gerror_t rbtree_max_value (rbtree_t ∗rbt, void ∗elem)
- gerror_t rbtree_find_node (rbtree_t ∗rbt, void ∗elem, rbnode_t ∗∗node)

4.5.1 Typedef Documentation

4.5.1.1 rbnode_t

```
typedef struct redblacknode_t rbnode_t
```

4.5.1.2 rbtree_compare_function

```
typedef int(* rbtree_compare_function) (void *a, void *b, void *arg)
```

4.5.1.3 rbtree_t

```
typedef struct redblacktree_t rbtree_t
```

4.5.1.4 redblacknode_t

```
typedef struct redblacknode_t redblacknode_t
```

4.5.1.5 redblacktree_t

```
typedef struct redblacktree_t redblacktree_t
```

4.5.2 Enumeration Type Documentation

4.5.2.1 rbcolor_t

```
enum rbcolor_t
```

Enumerator

G_RB_RED	
G_RB_BLACK	
G_RB_DOUBLE_BLACK	

4.5.2.2 rbcomp_t

```
enum rbcomp_t
```

Enumerator

G_RB_FIRST_IS_SMALLER	
G_RB_EQUAL	
G_RB_FIRST_IS_GREATER	

4.5.3 Function Documentation

4.5.3.1 rbtree_add()

```
gerror_t rbtree_add (
            rbtree_t * rbt,
            void * elem )
```

Add an element pointed by `elem` with size `rbt->member_size` in the rbtree.

Parameters

rbt	previous allocated rbtree_t structure
elem	pointer to the elem to be copied to the structure

Returns

GERROR_OK in case of sucess operation; GERROR_NULL_STRUCTURE in case `rbt` is null

4.5.3.2 rbtree_create()

```
gerror_t rbtree_create (
            rbtree_t * rbt,
            size_t member_size )
```

Populates the `rbt` structure and inicialize it. A red n black tree needs a rbtree_compare_function. The default function will only work for char, int and long. If you need a double or float you need to implement the compare function and set with the function `rbtree_set_compare_function`

Parameters

rbt	previous allocated rbtree_t struct
member_size	size in bytes of the indexed elements

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `rbt` is a NULL

4.5.3.3 rbtree_destroy()

```
gerror_t rbtree_destroy (
            rbtree_t * rbt )
```

Destroy (i.e. deallocates) the `rbt` structure fields.

Parameters

p	previous allocated pqueue_t struct

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `rbt` is a NULL

4.5.3.4 rbtree_find_node()

```
gerror_t rbtree_find_node (
            rbtree_t * rbt,
            void * elem,
            rbnode_t ** node )
```

Find a node with the value pointed by `elem` and write the pointer to `node`.

Parameters

rbt	previous allocated rbtree_t struct
elem	pointer element to find
node	pointer to the return node pointer; this pointer could be null

Returns

GERROR_OK in case of success operation; GERROR_NULL_ELEMENT_POINTER in case `elem` is null; GERROR_NULL_STRUCURE in case `rbt` is a NULL; By the end of the function, the `*node` will point to the found element or NULL otherwise

4.5.3.5 rbtree_max_node()

```
gerror_t rbtree_max_node (
            rbtree_t * rbt,
            rbnode_t ** node )
```

Find the maximal value in rbt and write the pointer to `node`.

Parameters

rbt	previous allocated rbtree_t struct
node	pointer to the return node pointer;

Returns

GERROR_OK in case of success operation; GERROR_EMPTY_STRUCTURE in case the `rbt` structure is empty GERROR_NULL_STRUCURE in case `rbt` is a NULL; GERROR_NULL_RETURN_POINTER in case that `node` By the end of the function, the `*node` will point to the found element or NULL otherwise

4.5.3.6 rbtree_max_value()

```
gerror_t rbtree_max_value (
            rbtree_t * rbt,
            void * elem )
```

Find the maximal value in rbt and write the value to `elem`.

Parameters

rbt	previous allocated rbtree_t struct
elem	pointer to a local in memory to write the maximal value

Returns

GERROR_OK in case of success operation; GERROR_EMPTY_STRUCTURE in case the `rbt` structure is empty GERROR_NULL_STRUCURE in case `rbt` is a NULL; GERROR_NULL_RETURN_POINTER in case that `node` By the end of the function, the `*node` will point to the found element or NULL otherwise

4.5.3.7 rbtree_min_node()

```
gerror_t rbtree_min_node (
            rbtree_t * rbt,
            rbnode_t ** node )
```

Find the minimal value in rbt and write the pointer to `node`.

Parameters

rbt	previous allocated rbtree_t struct
node	pointer to the return node pointer;

Returns

GERROR_OK in case of success operation; GERROR_EMPTY_STRUCTURE in case the `rbt` structure is empty GERROR_NULL_STRUCURE in case `rbt` is a NULL; GERROR_NULL_RETURN_POINTER in case that `node` By the end of the function, the `*node` will point to the found element or NULL otherwise

4.5.3.8 rbtree_min_value()

```
gerror_t rbtree_min_value (
            rbtree_t * rbt,
            void * elem )
```

Find the minimal value in rbt and write the value to `elem`.

Parameters

rbt	previous allocated rbtree_t struct
elem	pointer to a local in memory to write the minimal value

Returns

GERROR_OK in case of success operation; GERROR_EMPTY_STRUCTURE in case the `rbt` structure is empty GERROR_NULL_STRUCURE in case `rbt` is a NULL; GERROR_NULL_RETURN_POINTER in case that `node` By the end of the function, the `*node` will point to the found element or NULL otherwise

4.5.3.9 rbtree_remove_item()

```
gerror_t rbtree_remove_item (
            rbtree_t * rbt,
            void * elem )
```

Finds and remove the first element that match with `elem`.

Parameters

rbt	previous allocated rbtree_t structure
elem	pointer element to be removed

Returns

GERROR_OK in case of sucess operation; GERROR_NULL_STRUCTURE in case `rbt` is null; GERROR↩ _NULL_ELEMENT_POINTER in case `elem` is pointing to null; GERROR_TRY_REMOVE_EMPTY_STRU↩ CTURE in case the `rbt` structure is empty

4.5.3.10 rbtree_remove_node()

```
gerror_t rbtree_remove_node (
            rbtree_t * rbt,
            rbnode_t * node )
```

Removes the node `node` of the `rbt` tree.

Parameters

rbt	previous allocated rbtree_t structure
node	pointer to a `rbnode_t` structure.

Returns

GERROR_OK in case of sucess operation; GERROR_NULL_STRUCURE in case `rbt` is a NULL; GERR↩ OR_NULL_ELEMENT_POINTER in case `elem` is pointing to null; GERROR_TRY_REMOVE_EMPTY_ST↩ RUCTURE in case the `rbt` structure is empty

4.5.3.11 rbtree_set_compare_function()

```
gerror_t rbtree_set_compare_function (
            rbtree_t * rbt,
            rbtree_compare_function function,
            void * argument )
```

Change the default comparison function of the red n black tree `rbt` for `function` with the argument `argument`.

Parameters

rbt	pointer to a previous allocated `rbtree_t` structure
function	comparison function callback that has the following prototype: int compare(void* a, void* b) the a and b are the arguments returns -1 if `a` is smaller than b returns 0 if `a` is equal than `b` return 1 if `a` is bigger than `b`
argument	pointer to the argument to the comparison function

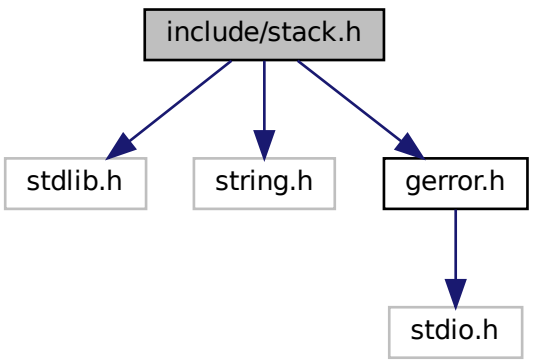
Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `t` is a NULL

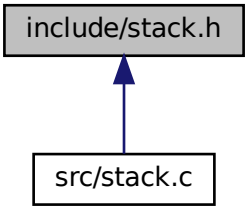
4.6 include/stack.h File Reference

```
#include <stdlib.h>
#include <string.h>
#include "gerror.h"
```

Include dependency graph for stack.h:



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



Classes

- struct snode_t
- struct stack_t

Typedefs

- typedef struct snode_t snode_t
- typedef struct stack_t stack_t

Functions

- gerror_t stack_create (struct stack_t *q, size_t member_size)
- gerror_t stack_push (struct stack_t *q, void *e)
- gerror_t stack_pop (struct stack_t *q, void *e)
- gerror_t stack_destroy (struct stack_t *q)

4.6.1 Typedef Documentation

4.6.1.1 snode_t

```
typedef struct snode_t snode_t
```

node of a stack

4.6.1.2 stack_t

```
typedef struct stack_t stack_t
```

represents the stack structure.

4.6.2 Function Documentation

4.6.2.1 stack_create()

```
gerror_t stack_create (
            struct stack_t * s,
            size_t member_size )
```

Creates a stack and populates the previous allocated structure pointed by `s`;

Parameters

s	pointer to a stack structure;
member_size	size of the elements that will be indexed by `s`

Returns

GERROR_OK in case of success operation; GERROR_NULL_ELEMENT in case that `e` is empty.

4.6.2.2 stack_destroy()

```
gerror_t stack_destroy (
            struct stack_t * s )
```

Deallocates the nodes of the structure pointed by `s`. This function WILL NOT deallocate the pointer `q`.

Parameters

s	pointer to a stack structure;

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `s` is a NULL

4.6.2.3 stack_pop()

```
gerror_t stack_pop (
            struct stack_t * s,
            void * e )
```

Pops the first element of the stack `s`.

Parameters

s	pointer to a stack structure;
e	pointer to the previous allocated element

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `s` is a NULL GERRO↩ R_NULL_HEAD in case that the head `s->head` GERROR_TRY_REMOVE_EMPTY_STRUCTURE in case that `s` is empty

4.6.2.4 stack_push()

```
gerror_t stack_push (
            struct stack_t * s,
            void * e )
```

Add the element `e` in the beginning of the stack `s`.

Parameters

s	pointer to a stack structure;
e	pointer to the element that will be indexed by s.

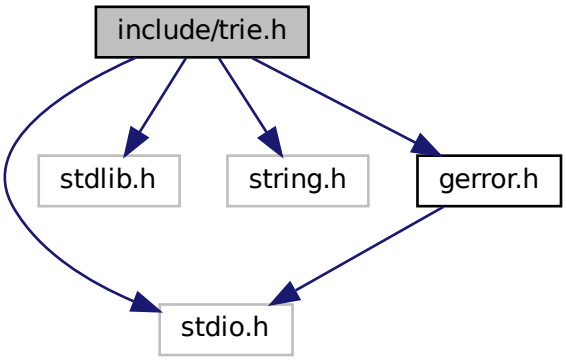
Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `s` is a NULL

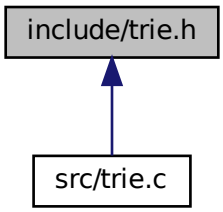
4.7 include/trie.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "gerror.h"
```

Include dependency graph for trie.h:



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



Classes

- struct tnode_t
- struct trie_t

Macros

- #define NBYTE (0x100)

Typedefs

- typedef struct tnode_t tnode_t
- typedef struct trie_t trie_t

Functions

- gerror_t trie_create (struct trie_t *t, size_t member_size)
- gerror_t trie_destroy (struct trie_t *t)
- gerror_t trie_add_element (struct trie_t *t, void *string, size_t size, void *elem)
- gerror_t trie_remove_element (struct trie_t *t, void *string, size_t size)
- gerror_t trie_get_element (struct trie_t *t, void *string, size_t size, void *elem)
- gerror_t trie_set_element (struct trie_t *t, void *string, size_t size, void *elem)
- tnode_t * trie_get_node_or_allocate (struct trie_t *t, void *string, size_t size)

4.7.1 Macro Definition Documentation

4.7.1.1 NBYTE

```
#define NBYTE (0x100)
```

4.7.2 Typedef Documentation

4.7.2.1 tnode_t

```
typedef struct tnode_t tnode_t
```

node of a trie_t element.

4.7.2.2 trie_t

```
typedef struct trie_t trie_t
```

Represents the trie structure.

4.7.3 Function Documentation

4.7.3.1 trie_add_element()

```
gerror_t trie_add_element (
            struct trie_t * t,
            void * string,
            size_t size,
            void * elem )
```

Adds the `elem` and maps it with the `string` with size `size`. This function overwrite any data left in the trie mapped with string.

Parameters

t	pointer to the trie structure;
string	pointer to the string of bytes to map elem;
size	size of the string of bytes
elem	pointer to the element to add

4.7.3.2 trie_create()

```
gerror_t trie_create (
            struct trie_t * t,
            size_t member_size )
```

Inicialize structure `t` with `member_size` size. The t has to be allocated.

Parameters

t	pointer to the allocated struct trie_t;
member_size	size in bytes of the indexed elements by the trie.

4.7.3.3 trie_destroy()

```
gerror_t trie_destroy (
            struct trie_t * t )
```

Destroy the members pointed by `t`. The structure is not freed.

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `t` is a NULL

4.7.3.4 trie_get_element()

```
gerror_t trie_get_element (
            struct trie_t * t,
            void * string,
            size_t size,
            void * elem )
```

Returns the element mapped by `string`. If the map does not exist, returns NULL.

Parameters

t	pointer to the structure;
string	pointer to the string of bytes to map elem;
size	size of the string of bytes.
elem	pointer to the memory allocated that will be write with the elem mapped by `string`

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `t` is a NULL

4.7.3.5 trie_get_node_or_allocate()

```
tnode_t* trie_get_node_or_allocate (
            struct trie_t * t,
            void * string,
            size_t size )
```

4.7.3.6 trie_remove_element()

```
gerror_t trie_remove_element (
            struct trie_t * t,
            void * string,
            size_t size )
```

Removes the element mapped by `string`.

Parameters

t	pointer to the structure trie_t;
string	pointer to the string of bytes to map elem;
size	size of the string of bytes.

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `t` is a NULL GERROR↩ _OUT_OF_BOUND the `elem` does not exist in `string map`

4.7.3.7 trie_set_element()

```
gerror_t trie_set_element (
            struct trie_t * t,
```

```
        void * string,
        size_t size,
        void * elem )
```

Sets the value mapped by `string`. Encapsulates the remove and add functions.

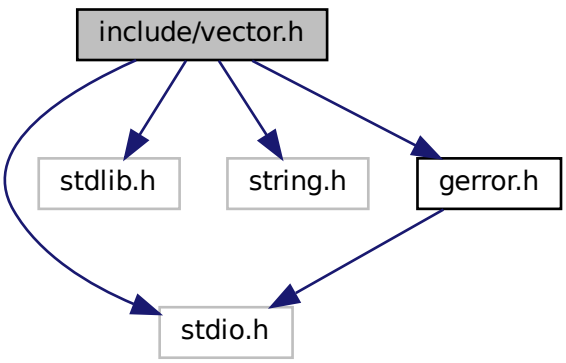
Parameters

t	pointer to the structure;
string	pointer to the string of bytes to map elem;
size	size of the string of bytes.
elem	pointer to the element to add

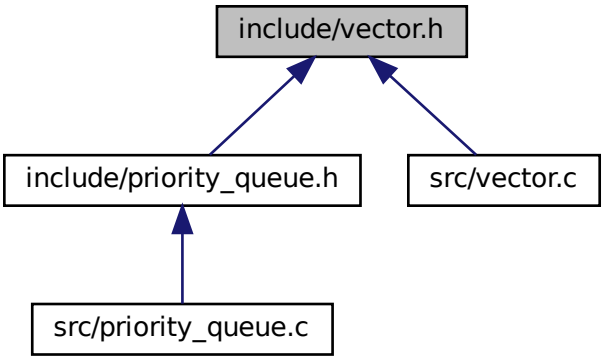
4.8 include/vector.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "gerror.h"
```

Include dependency graph for vector.h:



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



Classes

- struct vector_t

Typedefs

- typedef struct vector_t vector_t

Functions

- gerror_t vector_create (vector_t *v, size_t initial_size, size_t member_size)
- gerror_t vector_destroy (vector_t *v)
- gerror_t vector_resize_buffer (vector_t *v, size_t new_size)
- gerror_t vector_at (vector_t *v, size_t index, void *elem)
- void * vector_ptr_at (vector_t *v, size_t index)
- gerror_t vector_set_elem_at (vector_t *v, size_t index, void *elem)
- gerror_t vector_add (vector_t *v, void *elem)
- void vector_set_min_buf_siz (size_t new_min_buf_size)
- size_t vector_get_min_buf_siz (void)

4.8.1 Typedef Documentation

4.8.1.1 vector_t

```
typedef struct vector_t vector_t
```

4.8.2 Function Documentation

4.8.2.1 vector_add()

```
gerror_t vector_add (
            vector_t * v,
            void * elem )
```

adds the `elem` in the structure `vector_t` pointed by `v`.

Parameters

v	a pointer to `vector_t`
elem	the element to be add in `v`

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCTURE in case `v` is a NULL pointer

4.8.2.2 vector_at()

```
gerror_t vector_at (
            vector_t * v,
            size_t index,
            void * elem )
```

Get the element in the `index` position indexed by the `vector_t` structure pointed by `v`.

Parameters

v	a pointer to `vector_t`
index	index of the position
elem	pointer to a previous allocated memory that will receive the element

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `v` is a NULL pointer

4.8.2.3 vector_create()

```
gerror_t vector_create (
            vector_t * v,
```

```
            size_t initial_buf_siz,
            size_t member_size )
```

Populate the vetor_t structure pointed by v and allocates member_size*initial_size for initial buffer↩ _size.

Parameters

v	a pointer to vector_t structure already allocated;
inicial_buf_size	number of the members of the initial allocated buffer;
member_size	size of every member indexed by v.

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case v is a NULL pointer

4.8.2.4 vector_destroy()

```
gerror_t vector_destroy (
            vector_t * v )
```

Destroy the structure vector_t pointed by v.

Parameters

v	a pointer to vector_t structure

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case v is a NULL pointer

4.8.2.5 vector_get_min_buf_siz()

```
size_t vector_get_min_buf_siz (
            void  )
```

Returns the vector_min_siz: a private variable that holds the minimal number of elements that vector_t will index. This variable is important for avoid multiple small resizes in the vector_t container.

Returns

vector_min_siz

4.8.2.6 vector_ptr_at()

```
void* vector_ptr_at (
            vector_t * v,
            size_t index )
```

Calculate the pointer at `index` position.

Parameters

v	a pointer to `vector_t`
index	index of the pointer

Returns

a pointer to the `index` element NULL in case of out of bound

4.8.2.7 vector_resize_buffer()

```
gerror_t vector_resize_buffer (
            vector_t * v,
            size_t n_elements )
```

Resize the buffer in the `vector_t` strucuture pointed by `v`.

Parameters

v	a pointer to `vector_t` structure.
new_size	the new size of the `v`

4.8.2.8 vector_set_elem_at()

```
gerror_t vector_set_elem_at (
            vector_t * v,
            size_t index,
            void * elem )
```

set the element at `index` pointed by `v` with the element pointed by `elem`.

Parameters

v	a pointer to `vector_t`
index	index of the position
elem	the element to be set in `v`

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `v` is a NULL pointer

4.8.2.9 vector_set_min_buf_siz()

```
void vector_set_min_buf_siz (
            size_t new_min_buf_siz )
```

Set the `vector_min_siz`: a private variable that holds the minimal number of elements that `vector_t` will index. This variable is important for avoid multiple small resizes in the `vector_t` container.

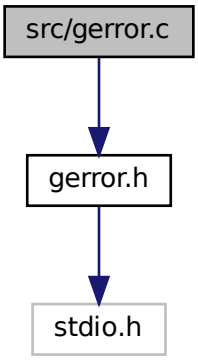
Parameters

new_min_buf_siz	the new size of `vector_min_siz`

4.9 src/gerror.c File Reference

```
#include "gerror.h"
```

Include dependency graph for gerror.c:



Functions

- char * gerror_to_str (gerror_t g)

Variables

- char * gerror_to_string [GERROR_N_ERROR]

4.9.1 Function Documentation

4.9.1.1 gerror_to_str()

```
char* gerror_to_str (
            gerror_t g )
```

4.9.2 Variable Documentation

4.9.2.1 gerror_to_string

```
char* gerror_to_string[GERROR_N_ERROR]
```

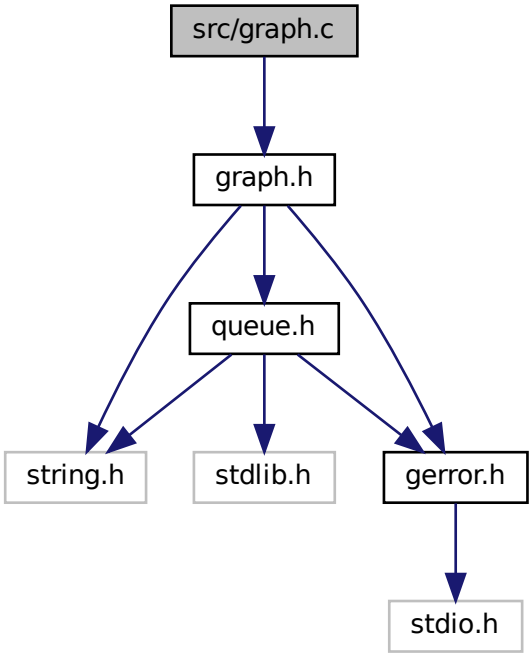
Initial value:

```
= {
    "Success",
    "Null pointer to structure",
    "Null pointer to the head of structure",
    "Null pointer to the node",
    "Null pointer passed to write the return data",
    "Attempt to pass a empty structure",
    "Attempt to remove an element but the structure is empty",
    "Attempt to add a edge with inexistent vertex",
    "Attempt to access a position out of the container or buffer",
    "Attempt to operate a function with a prohibitive null pointer element",
    "Attempt to remove an element that is not in the structure",
    "Element not found"
}
```

4.10 src/graph.c File Reference

```
#include "graph.h"
```

Include dependency graph for graph.c:



Functions

- gerror_t graph_create (graph_t *g, size_t size, size_t member_size)
- gerror_t graph_add_edge (graph_t *g, size_t from, size_t to)
- gerror_t graph_get_label_at (graph_t *g, size_t index, void *label)
- gerror_t graph_set_label_at (graph_t *g, size_t index, void *label)
- gerror_t graph_destroy (graph_t *g)

4.10.1 Function Documentation

4.10.1.1 graph_add_edge()

```
gerror_t graph_add_edge (
            graph_t * g,
            size_t from,
            size_t to )
```

Adds an edge on the graph `g` from the vertex `from` to the vertex `to`. Where `from` and `to` are indexes of these vertex.

Parameters

g	pointer to a graph structure;
from	index of the first vertex;
to	index of the incident vertex.

Returns

GERROR_OK in case of success operation; GERROR_TRY_ADD_EDGE_NO_VERTEX in case that `from` or `to` not exists in the graph

4.10.1.2 graph_create()

```
gerror_t graph_create (
            graph_t * g,
            size_t size,
            size_t member_size )
```

Creates a graph and populates the previous allocated structure pointed by `g`;

Parameters

g	pointer to a graph structure;
member_size	size of the elements that will be indexed by `g`

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `g` is a NULL

4.10.1.3 graph_destroy()

```
gerror_t graph_destroy (
            graph_t * g )
```

Deallocates the structures in `g`. This function WILL NOT deallocate the pointer `g`.

Parameters

g	pointer to a graph structure;

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `g` is a NULL

4.10.1.4 graph_get_label_at()

```
gerror_t graph_get_label_at (
            graph_t * g,
            size_t index,
            void * label )
```

Gets the label of the vertex in the `index` position of the graph `g`.

Parameters

g	pointer to a graph structure;
index	index of the vertex;
label	pointer to the memory allocated that will be write with the label in `index`

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `g` is a NULL

4.10.1.5 graph_set_label_at()

```
gerror_t graph_set_label_at (
            graph_t * g,
            size_t index,
            void * label )
```

Sets the label at the `index` to `label`.

Parameters

g	pointer to a graph structure;
index	index of the vertex;
label	the new label of the vertex positioned in `index`

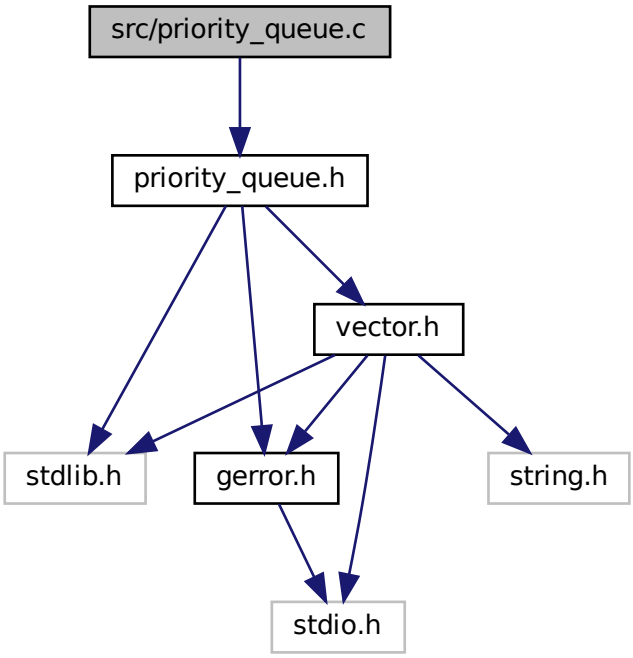
Returns

GERROR_OK in case of success operation; GERROR_ACCESS_OUT_OF_BOUND in case that `index` is out of bound

4.11 src/priority_queue.c File Reference

```
#include "priority_queue.h"
```

Include dependency graph for priority_queue.c:



Macros

- #define PARENT(i) ((i-1)/2)
- #define LEFT(i) (((i+1)∗2)-1)
- #define RIGHT(i) (LEFT(i)+1)

Functions

- void nswap (void ∗a, void ∗b, size_t n)
- int pqueue_default_compare_function (void ∗a, void ∗b, void ∗arg)
- void max_heapify (pqueue_t ∗p, size_t i)
- gerror_t pqueue_create (pqueue_t ∗p, size_t member_size)
- gerror_t pqueue_destroy (pqueue_t ∗p)
- gerror_t pqueue_set_compare_function (pqueue_t ∗p, pqueue_compare_function function, void ∗argument)
- gerror_t pqueue_add (pqueue_t ∗p, void ∗e)
- gerror_t pqueue_max_priority (pqueue_t ∗p, void ∗e)
- gerror_t pqueue_extract (pqueue_t ∗p, void ∗e)

4.11.1 Macro Definition Documentation

4.11.1.1 LEFT

```
#define LEFT(
              i ) (((i+1)*2)-1)
```

4.11.1.2 PARENT

```
#define PARENT(
              i ) ((i-1)/2)
```

4.11.1.3 RIGHT

```
#define RIGHT(
              i ) (LEFT(i)+1)
```

4.11.2 Function Documentation

4.11.2.1 max_heapify()

```
void max_heapify (
             pqueue_t * p,
             size_t i )
```

4.11.2.2 nswap()

```
void nswap (
             void * a,
             void * b,
             size_t n )
```

4.11.2.3 pqueue_add()

```
gerror_t pqueue_add (
             pqueue_t * p,
             void * e )
```

Adds an element in the queue and max heap the queue. TODO: A more datailed description of pqueue_add.

Parameters

p	previous allocated pqueue_t struct
e	the element to be added

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `t` is a NULL

4.11.2.4 pqueue_create()

```
gerror_t pqueue_create (
            pqueue_t * p,
            size_t member_size )
```

Populates the `p` structure and inicialize it. A priority queue needs a pqueue_compare_function. The default function will only work for char, int and long. If you need a double or float you need to implement the compare function and set with the function `pqueue_set_compare_function`

Parameters

p	previous allocated pqueue_t struct
member_size	size in bytes of the indexed elements

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `p` is a NULL

4.11.2.5 pqueue_default_compare_function()

```
int pqueue_default_compare_function (
            void * a,
            void * b,
            void * arg )
```

4.11.2.6 pqueue_destroy()

```
gerror_t pqueue_destroy (
            pqueue_t * p )
```

Destroy (i.e. deallocates) the `p` structure fields. TODO: A more detailed description of pqueue_destroy.

Parameters

p	previous allocated pqueue_t struct

Returns

TODO

4.11.2.7 pqueue_extract()

```
gerror_t pqueue_extract (
            pqueue_t * p,
            void * e )
```

Extracts the highest priority element in the queue and writes in `e` pointer.

Parameters

p	previous allocated pqueue_t struct
e	pointer to previous allocated variable

Returns

GERROR_OK in case of success operation; GERROR_ACESS_OUT_OF_BOUND in case the queue is empty GERROR_NULL_STRUCURE in case `t` is a NULL

4.11.2.8 pqueue_max_priority()

```
gerror_t pqueue_max_priority (
            pqueue_t * p,
            void * e )
```

Returns and does not remove the highest priority of the queue. TODO: A more datailed description of pqueue_↩ max_priority.

Parameters

p	previous allocated pqueue_t struct
e	pointer to previous allocated variable with `member_size` size that will receive a copy of the highest priority element of the queue.

Returns

GERROR_OK in case of success operation; GERROR_ACESS_OUT_OF_BOUND in case the queue is empty GERROR_NULL_STRUCURE in case `t` is a NULL

4.11.2.9 pqueue_set_compare_function()

```
gerror_t pqueue_set_compare_function (
            pqueue_t * p,
            pqueue_compare_function function,
            void * argument )
```

Change the default comparison function of the priority queue `p` by `function` with the argument `argument`.

Parameters

p	previous allocated pqueue_t struct
function	comparison function callback that has the following prototype: int compare(void* a, void* b) the a and b are the arguments returns -1 if `a` has priority BIG than `b` returns 0 if `a` has priority EQUAL than `b` return 1 if `a` has priority LESS than `b`
argument	pointer to the argument to the comparison function

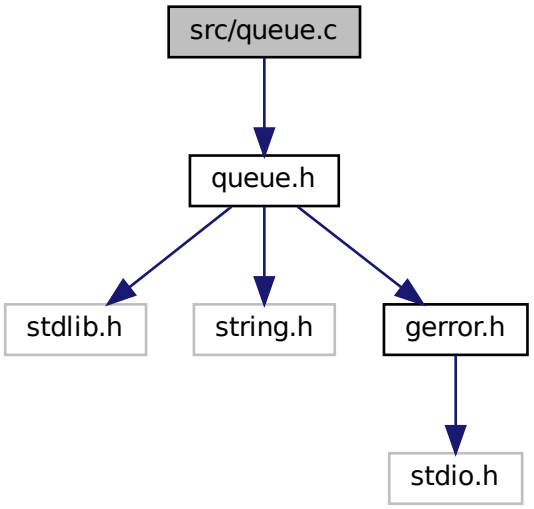
Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `t` is a NULL

4.12 src/queue.c File Reference

```
#include "queue.h"
```

Include dependency graph for queue.c:



Functions

- gerror_t queue_create (struct queue_t *q, size_t member_size)
- gerror_t queue_enqueue (struct queue_t *q, void *e)
- gerror_t queue_dequeue (struct queue_t *q, void *e)
- gerror_t queue_remove (struct queue_t *q, struct qnode_t *node, void *e)
- gerror_t queue_destroy (struct queue_t *q)

4.12.1 Function Documentation

4.12.1.1 queue_create()

```
gerror_t queue_create (
            struct queue_t * q,
            size_t member_size )
```

Creates a queue and populates the previous allocated structure pointed by `q`;

Parameters

q	pointer to a queue structure;
member_size	size of the elements that will be indexed by `q`

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `q` is a NULL pointer

4.12.1.2 queue_dequeue()

```
gerror_t queue_dequeue (
            struct queue_t * q,
            void * e )
```

Dequeues the first element of the queue `q`

Parameters

q	pointer to a queue structure;
e	pointer to the previous allocated element memory that will be write with de dequeued element.

Returns

GERROR_OK in case of success operation; GERROR_NULL_HEAD in case that the head `q->head` is a null pointer. GERROR_NULL_STRUCURE in case `q` is a NULL pointer GERROR_TRY_REMOVE_EMPT↩ Y_STRUCTURE in case that `q` has no element.

4.12.1.3 queue_destroy()

```
gerror_t queue_destroy (
            struct queue_t * q )
```

Deallocate the nodes of the queue q. This function WILL NOT deallocate the pointer q.

Parameters

q	pointer to a queue structure;

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `q` is a NULL pointer

4.12.1.4 queue_enqueue()

```
gerror_t queue_enqueue (
            struct queue_t * q,
            void * e )
```

Enqueues the element pointed by `e` in the queue `q`.

Parameters

q	pointer to a queue structure;
e	pointer to the element that will be indexed by q.

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `q` is a NULL pointer

4.12.1.5 queue_remove()

```
gerror_t queue_remove (
            struct queue_t * q,
            struct qnode_t * node,
            void * e )
```

Removes the element `node` of the queue `q`.

Parameters

q	pointer to a queue structure;
node	element to be removed from the queue
e	pointer to the memory that will be write with the removed element

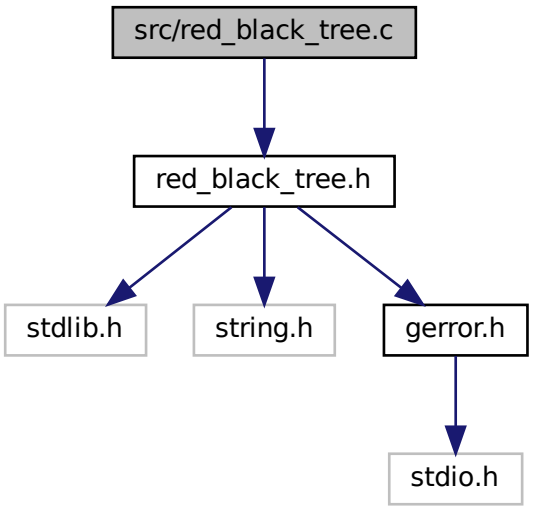
Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `q` is a NULL pointer GERROR_NULL_NODE in case `node` is NULL; GERROR_TRY_REMOVE_EMPTY_STRUCTURE in case that `q` has no element.

4.13 src/red_black_tree.c File Reference

```
#include "red_black_tree.h"
```

Include dependency graph for red_black_tree.c:



Enumerations

- enum rbc_t { RB_LEFT, RB_RIGHT }

Functions

- rbnode_t * rbnode_destroy (rbnode_t *node)
- void rbtree_insert_fixup (rbtree_t *rbt, rbnode_t *node)
- void fix_insert_case (rbtree_t *rbt, rbnode_t **node, rbnode_t *uncle, int c, int l)
- rbnode_t * create_node (rbtree_t *rbt, void *elem)

- void left_rotate (rbtree_t *rbt, rbnode_t *node)
- void right_rotate (rbtree_t *rbt, rbnode_t *node)
- void rbtree_transplant (rbtree_t *rbt, rbnode_t *u, rbnode_t *v)
- void rbtree_delete_fixup (rbtree_t *rbt, rbnode_t *node)
- int rbtree_identify_case (rbnode_t *node, rbc_t *side)
- rbnode_t * rbtree_create_double_black ()
- rbnode_t * rbtree_find_minimal_node (rbnode_t *node)
- void rbtree_remove_double_black (rbtree_t *rbt, rbnode_t *db)
- int rbnode_is_red (rbnode_t *node)
- int rbnode_is_black (rbnode_t *node)
- int rbtree_default_compare_function (void *a, void *b, void *arg)
- gerror_t rbtree_create (rbtree_t *rbt, size_t member_size)
- gerror_t rbtree_destroy (rbtree_t *rbt)
- gerror_t rbtree_set_compare_function (rbtree_t *rbt, rbtree_compare_function function, void *argument)
- gerror_t rbtree_add (rbtree_t *rbt, void *elem)
- gerror_t rbtree_remove_item (rbtree_t *rbt, void *elem)
- gerror_t rbtree_remove_node (rbtree_t *rbt, rbnode_t *node)
- gerror_t rbtree_find_node (rbtree_t *rbt, void *elem, rbnode_t **node)
- gerror_t rbtree_min_node (rbtree_t *rbt, rbnode_t **node)
- gerror_t rbtree_max_node (rbtree_t *rbt, rbnode_t **node)
- gerror_t rbtree_min_value (rbtree_t *rbt, void *elem)
- gerror_t rbtree_max_value (rbtree_t *rbt, void *elem)

4.13.1 Enumeration Type Documentation

4.13.1.1 rbc_t

```
enum rbc_t
```

Enumerator

RB_LEFT	
RB_RIGHT	

4.13.2 Function Documentation

4.13.2.1 create_node()

```
rbnode_t * create_node (
            rbtree_t * rbt,
            void * elem )
```

4.13.2.2 fix_insert_case()

```
void fix_insert_case (
             rbtree_t * rbt,
             rbnode_t ** node,
             rbnode_t * uncle,
             int c,
             int l )
```

4.13.2.3 left_rotate()

```
void left_rotate (
             rbtree_t * rbt,
             rbnode_t * node )
```

4.13.2.4 rbnode_destroy()

```
rbnode_t * rbnode_destroy (
             rbnode_t * node )
```

4.13.2.5 rbnode_is_black()

```
int rbnode_is_black (
             rbnode_t * node )
```

4.13.2.6 rbnode_is_red()

```
int rbnode_is_red (
             rbnode_t * node )
```

4.13.2.7 rbtree_add()

```
gerror_t rbtree_add (
             rbtree_t * rbt,
             void * elem )
```

Add an element pointed by `elem` with size `rbt->member_size` in the rbtree.

Parameters

rbt	previous allocated rbtree_t structure
elem	pointer to the elem to be copied to the structure

Returns

GERROR_OK in case of sucess operation; GERROR_NULL_STRUCTURE in case `rbt` is null

4.13.2.8 rbtree_create()

```
gerror_t rbtree_create (
            rbtree_t * rbt,
            size_t member_size )
```

Populates the `rbt` structure and inicialize it. A red n black tree needs a rbtree_compare_function. The default function will only work for char, int and long. If you need a double or float you need to implement the compare function and set with the function `rbtree_set_compare_function`

Parameters

rbt	previous allocated rbtree_t struct
member_size	size in bytes of the indexed elements

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `rbt` is a NULL

4.13.2.9 rbtree_create_double_black()

```
rbnode_t * rbtree_create_double_black ( )
```

4.13.2.10 rbtree_default_compare_function()

```
int rbtree_default_compare_function (
            void * a,
            void * b,
            void * arg )
```

4.13.2.11 rbtree_delete_fixup()

```
void rbtree_delete_fixup (
              rbtree_t * rbt,
              rbnode_t * node )
```

4.13.2.12 rbtree_destroy()

```
gerror_t rbtree_destroy (
              rbtree_t * rbt )
```

Destroy (i.e. deallocates) the `rbt` structure fields.

Parameters

p	previous allocated pqueue_t struct

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `rbt` is a NULL

4.13.2.13 rbtree_find_minimal_node()

```
rbnode_t * rbtree_find_minimal_node (
              rbnode_t * node )
```

4.13.2.14 rbtree_find_node()

```
gerror_t rbtree_find_node (
              rbtree_t * rbt,
              void * elem,
              rbnode_t ** node )
```

Find a node with the value pointed by `elem` and write the pointer to `node`.

Parameters

rbt	previous allocated rbtree_t struct
elem	pointer element to find
node	pointer to the return node pointer; this pointer could be null

Returns

GERROR_OK in case of success operation; GERROR_NULL_ELEMENT_POINTER in case `elem` is null; GERROR_NULL_STRUCURE in case `rbt` is a NULL; By the end of the function, the `*node` will point to the found element or NULL otherwise

4.13.2.15 rbtree_identify_case()

```
int rbtree_identify_case (
            rbnode_t * node,
            rbc_t * side )
```

4.13.2.16 rbtree_insert_fixup()

```
void rbtree_insert_fixup (
            rbtree_t * rbt,
            rbnode_t * node )
```

4.13.2.17 rbtree_max_node()

```
gerror_t rbtree_max_node (
            rbtree_t * rbt,
            rbnode_t ** node )
```

Find the maximal value in rbt and write the pointer to `node`.

Parameters

rbt	previous allocated rbtree_t struct
node	pointer to the return node pointer;

Returns

GERROR_OK in case of success operation; GERROR_EMPTY_STRUCTURE in case the `rbt` structure is empty GERROR_NULL_STRUCURE in case `rbt` is a NULL; GERROR_NULL_RETURN_POINTER in case that `node` By the end of the function, the `*node` will point to the found element or NULL otherwise

4.13.2.18 rbtree_max_value()

```
gerror_t rbtree_max_value (
            rbtree_t * rbt,
            void * elem )
```

Find the maximal value in rbt and write the value to `elem`.

Parameters

rbt	previous allocated rbtree_t struct
elem	pointer to a local in memory to write the maximal value

Returns

GERROR_OK in case of success operation; GERROR_EMPTY_STRUCTURE in case the `rbt` structure is empty GERROR_NULL_STRUCURE in case `rbt` is a NULL; GERROR_NULL_RETURN_POINTER in case that `node` By the end of the function, the `*node` will point to the found element or NULL otherwise

4.13.2.19 rbtree_min_node()

```
gerror_t rbtree_min_node (
            rbtree_t * rbt,
            rbnode_t ** node )
```

Find the minimal value in rbt and write the pointer to `node`.

Parameters

rbt	previous allocated rbtree_t struct
node	pointer to the return node pointer;

Returns

GERROR_OK in case of success operation; GERROR_EMPTY_STRUCTURE in case the `rbt` structure is empty GERROR_NULL_STRUCURE in case `rbt` is a NULL; GERROR_NULL_RETURN_POINTER in case that `node` By the end of the function, the `*node` will point to the found element or NULL otherwise

4.13.2.20 rbtree_min_value()

```
gerror_t rbtree_min_value (
            rbtree_t * rbt,
            void * elem )
```

Find the minimal value in rbt and write the value to `elem`.

Parameters

rbt	previous allocated rbtree_t struct
elem	pointer to a local in memory to write the minimal value

Returns

GERROR_OK in case of success operation; GERROR_EMPTY_STRUCTURE in case the `rbt` structure is empty GERROR_NULL_STRUCURE in case `rbt` is a NULL; GERROR_NULL_RETURN_POINTER in case that `node` By the end of the function, the `*node` will point to the found element or NULL otherwise

4.13.2.21 rbtree_remove_double_black()

```
void rbtree_remove_double_black (
            rbtree_t * rbt,
            rbnode_t * db )
```

4.13.2.22 rbtree_remove_item()

```
gerror_t rbtree_remove_item (
            rbtree_t * rbt,
            void * elem )
```

Finds and remove the first element that match with `elem`.

Parameters

rbt	previous allocated rbtree_t structure
elem	pointer element to be removed

Returns

GERROR_OK in case of sucess operation; GERROR_NULL_STRUCTURE in case `rbt` is null; GERROR↩ _NULL_ELEMENT_POINTER in case `elem` is pointing to null; GERROR_TRY_REMOVE_EMPTY_STRU↩ CTURE in case the `rbt` structure is empty

4.13.2.23 rbtree_remove_node()

```
gerror_t rbtree_remove_node (
            rbtree_t * rbt,
            rbnode_t * node )
```

Removes the node `node` of the `rbt` tree.

Parameters

rbt	previous allocated rbtree_t structure
node	pointer to a `rbnode_t` structure.

Returns

GERROR_OK in case of sucess operation; GERROR_NULL_STRUCURE in case `rbt` is a NULL; GERR↩ OR_NULL_ELEMENT_POINTER in case `elem` is pointing to null; GERROR_TRY_REMOVE_EMPTY_ST↩ RUCTURE in case the `rbt` structure is empty

4.13.2.24 rbtree_set_compare_function()

```
gerror_t rbtree_set_compare_function (
            rbtree_t * rbt,
            rbtree_compare_function function,
            void * argument )
```

Change the default comparison function of the red n black tree `rbt` for `function` with the argument `argument`.

Parameters

rbt	pointer to a previous allocated `rbtree_t` structure
function	comparison function callback that has the following prototype: int compare(void* a, void* b) the a and b are the arguments returns -1 if `a` is smaller than b returns 0 if `a` is equal than `b` return 1 if `a` is bigger than `b`
argument	pointer to the argument to the comparison function

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `t` is a NULL

4.13.2.25 rbtree_transplant()

```
void rbtree_transplant (
            rbtree_t * rbt,
            rbnode_t * u,
            rbnode_t * v )
```

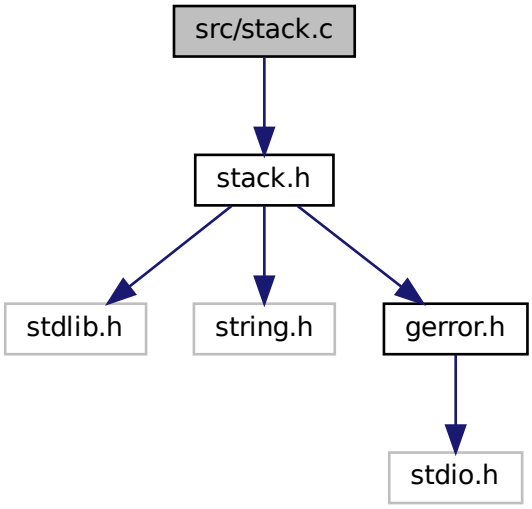
4.13.2.26 right_rotate()

```
void right_rotate (
            rbtree_t * rbt,
            rbnode_t * node )
```

4.14 src/stack.c File Reference

```
#include "stack.h"
```

Include dependency graph for stack.c:



Functions

- gerror_t stack_create (struct stack_t *s, size_t member_size)
- gerror_t stack_push (struct stack_t *s, void *e)
- gerror_t stack_pop (struct stack_t *s, void *e)
- gerror_t stack_destroy (struct stack_t *s)

4.14.1 Function Documentation

4.14.1.1 stack_create()

```
gerror_t stack_create (
            struct stack_t * s,
            size_t member_size )
```

Creates a stack and populates the previous allocated structure pointed by `s`;

Parameters

s	pointer to a stack structure;
member_size	size of the elements that will be indexed by `s`

Returns

GERROR_OK in case of success operation; GERROR_NULL_ELEMENT in case that e is empty.

4.14.1.2 stack_destroy()

```
gerror_t stack_destroy (
            struct stack_t * s )
```

Deallocates the nodes of the structure pointed by s. This function WILL NOT deallocate the pointer q.

Parameters

s	pointer to a stack structure;

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case s is a NULL

4.14.1.3 stack_pop()

```
gerror_t stack_pop (
            struct stack_t * s,
            void * e )
```

Pops the first element of the stack s.

Parameters

s	pointer to a stack structure;
e	pointer to the previous allocated element

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case s is a NULL GERRO↩ R_NULL_HEAD in case that the head s->head GERROR_TRY_REMOVE_EMPTY_STRUCTURE in case that s is empty

4.14.1.4 stack_push()

```
gerror_t stack_push (
            struct stack_t * s,
            void * e )
```

Add the element e in the beginning of the stack s.

Parameters

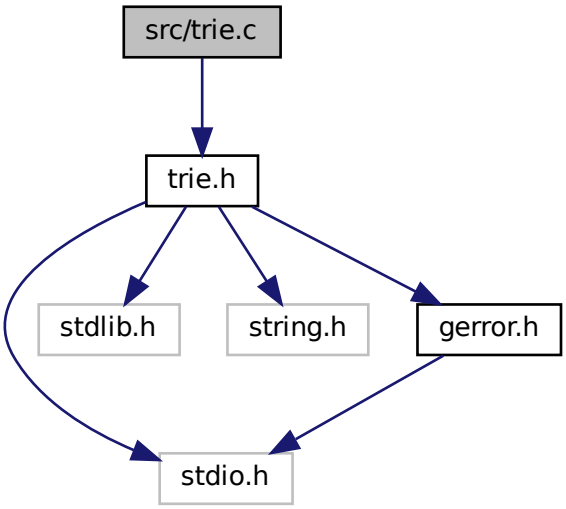
s	pointer to a stack structure;
e	pointer to the element that will be indexed by s.

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `s` is a NULL

4.15 src/trie.c File Reference

`#include "trie.h"`
Include dependency graph for trie.c:



Functions

- tnode_t * trie_get_node_or_allocate (struct trie_t *t, void *string, size_t size)
- tnode_t * node_at (struct trie_t *t, void *string, size_t size)
- gerror_t trie_create (struct trie_t *t, size_t member_size)
- void trie_destroy_tnode (struct tnode_t *node)
- gerror_t trie_destroy (struct trie_t *t)
- gerror_t trie_add_element (struct trie_t *t, void *string, size_t size, void *elem)
- gerror_t trie_remove_element (struct trie_t *t, void *string, size_t size)
- gerror_t trie_get_element (struct trie_t *t, void *string, size_t size, void *elem)
- gerror_t trie_set_element (struct trie_t *t, void *string, size_t size, void *elem)

4.15.1 Function Documentation

4.15.1.1 node_at()

```
tnode_t* node_at (
            struct trie_t * t,
            void * string,
            size_t size )
```

4.15.1.2 trie_add_element()

```
gerror_t trie_add_element (
            struct trie_t * t,
            void * string,
            size_t size,
            void * elem )
```

Adds the `elem` and maps it with the `string` with size `size`. This function overwrite any data left in the trie mapped with string.

Parameters

t	pointer to the trie structure;
string	pointer to the string of bytes to map elem;
size	size of the string of bytes
elem	pointer to the element to add

4.15.1.3 trie_create()

```
gerror_t trie_create (
            struct trie_t * t,
            size_t member_size )
```

Inicialize structure `t` with `member_size` size. The t has to be allocated.

Parameters

t	pointer to the allocated struct trie_t;
member_size	size in bytes of the indexed elements by the trie.

4.15.1.4 trie_destroy()

```
gerror_t trie_destroy (
            struct trie_t * t )
```

Destroy the members pointed by `t`. The structure is not freed.

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `t` is a NULL

4.15.1.5 trie_destroy_tnode()

```
void trie_destroy_tnode (
            struct tnode_t * node )
```

4.15.1.6 trie_get_element()

```
gerror_t trie_get_element (
            struct trie_t * t,
            void * string,
            size_t size,
            void * elem )
```

Returns the element mapped by `string`. If the map does not exist, returns NULL.

Parameters

t	pointer to the structure;
string	pointer to the string of bytes to map elem;
size	size of the string of bytes.
elem	pointer to the memory allocated that will be write with the elem mapped by `string`

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `t` is a NULL

4.15.1.7 trie_get_node_or_allocate()

```
tnode_t* trie_get_node_or_allocate (
            struct trie_t * t,
            void * string,
            size_t size )
```

4.15.1.8 trie_remove_element()

```
gerror_t trie_remove_element (
            struct trie_t * t,
            void * string,
            size_t size )
```

Removes the element mapped by `string`.

Parameters

t	pointer to the structure trie_t;
string	pointer to the string of bytes to map elem;
size	size of the string of bytes.

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `t` is a NULL GERROR↩ _OUT_OF_BOUND the `elem` does not exist in `string map`

4.15.1.9 trie_set_element()

```
gerror_t trie_set_element (
            struct trie_t * t,
            void * string,
            size_t size,
            void * elem )
```

Sets the value mapped by `string`. Encapsulates the remove and add functions.

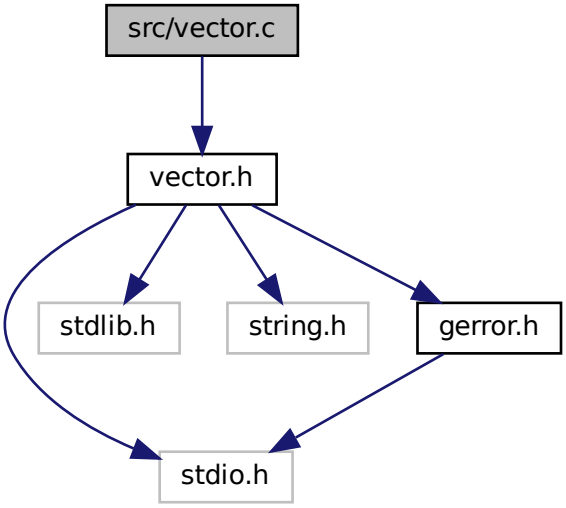
Parameters

t	pointer to the structure;
string	pointer to the string of bytes to map elem;
size	size of the string of bytes.
elem	pointer to the element to add

4.16 src/vector.c File Reference

```
#include "vector.h"
```

Include dependency graph for vector.c:



Macros

- #define VECTOR_MIN_SIZ 8

Functions

- gerror_t vector_create (vector_t *v, size_t initial_buf_siz, size_t member_size)
- gerror_t vector_destroy (vector_t *v)
- size_t vector_get_min_buf_siz (void)
- void vector_set_min_buf_siz (size_t new_min_buf_siz)
- gerror_t vector_resize_buffer (vector_t *v, size_t n_elements)
- gerror_t vector_at (vector_t *v, size_t index, void *elem)
- gerror_t vector_set_elem_at (vector_t *v, size_t index, void *elem)
- gerror_t vector_add (vector_t *v, void *elem)
- void * vector_ptr_at (vector_t *v, size_t index)

Variables

- size_t vector_min_siz = VECTOR_MIN_SIZ

4.16.1 Macro Definition Documentation

4.16.1.1 VECTOR_MIN_SIZ

```
#define VECTOR_MIN_SIZ 8
```

4.16.2 Function Documentation

4.16.2.1 vector_add()

```
gerror_t vector_add (
            vector_t * v,
            void * elem )
```

adds the `elem` in the structure `vector_t` pointed by `v`.

Parameters

v	a pointer to `vector_t`
elem	the element to be add in `v`

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCTURE in case `v` is a NULL pointer

4.16.2.2 vector_at()

```
gerror_t vector_at (
            vector_t * v,
            size_t index,
            void * elem )
```

Get the element in the `index` position indexed by the `vector_t` structure pointed by `v`.

Parameters

v	a pointer to `vector_t`
index	index of the position
elem	pointer to a previous allocated memory that will receive the element

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `v` is a NULL pointer

4.16.2.3 vector_create()

```
gerror_t vector_create (
            vector_t * v,
            size_t initial_buf_siz,
            size_t member_size )
```

Populate the `vetor_t` structure pointed by `v` and allocates `member_size*initial_size` for initial buffer_size.

Parameters

v	a pointer to `vector_t` structure already allocated;
inicial_buf_size	number of the members of the initial allocated buffer;
member_size	size of every member indexed by `v`.

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `v` is a NULL pointer

4.16.2.4 vector_destroy()

```
gerror_t vector_destroy (
            vector_t * v )
```

Destroy the structure `vector_t` pointed by `v`.

Parameters

v	a pointer to `vector_t` structure

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `v` is a NULL pointer

4.16.2.5 vector_get_min_buf_siz()

```
size_t vector_get_min_buf_siz (
            void  )
```

Returns the `vector_min_siz`: a private variable that holds the minimal number of elements that `vector_t` will index. This variable is important for avoid multiple small resizes in the `vector_t` container.

Returns

vector_min_siz

4.16.2.6 vector_ptr_at()

```
void* vector_ptr_at (
            vector_t * v,
            size_t index )
```

Calculate the pointer at `index` position.

Parameters

v	a pointer to `vector_t`
index	index of the pointer

Returns

a pointer to the `index` element NULL in case of out of bound

4.16.2.7 vector_resize_buffer()

```
gerror_t vector_resize_buffer (
            vector_t * v,
            size_t n_elements )
```

Resize the buffer in the `vector_t` strucuture pointed by `v`.

Parameters

v	a pointer to `vector_t` structure.
new_size	the new size of the `v`

4.16.2.8 vector_set_elem_at()

```
gerror_t vector_set_elem_at (
            vector_t * v,
            size_t index,
            void * elem )
```

set the element at `index` pointed by `v` with the element pointed by `elem`.

Parameters

v	a pointer to `vector_t`
index	index of the position
elem	the element to be set in `v`

Returns

GERROR_OK in case of success operation; GERROR_NULL_STRUCURE in case `v` is a NULL pointer

4.16.2.9 vector_set_min_buf_siz()

```
void vector_set_min_buf_siz (
            size_t new_min_buf_siz )
```

Set the `vector_min_siz`: a private variable that holds the minimal number of elements that `vector_t` will index. This variable is important for avoid multiple small resizes in the `vector_t` container.

Parameters

new_min_buf_siz	the new size of `vector_min_siz`

4.16.3 Variable Documentation

4.16.3.1 vector_min_siz

```
size_t vector_min_siz = VECTOR_MIN_SIZ
```

Index

adj
    graph_t, 6

buffer_size
    vector_t, 18

children
    tnode_t, 16
color
    redblacknode_t, 11
compare
    priority_queue_t, 7
    redblacktree_t, 12
compare_argument
    priority_queue_t, 7
    redblacktree_t, 12
create_node
    red_black_tree.c, 67

data
    qnode_t, 8
    redblacknode_t, 11
    snode_t, 14
    vector_t, 18

E
    graph_t, 6

fix_insert_case
    red_black_tree.c, 67

gerror.c
    gerror_to_str, 55
    gerror_to_string, 55
gerror.h
    gerror_t, 20
    gerror_to_str, 21
gerror_t
    gerror.h, 20
gerror_to_str
    gerror.c, 55
    gerror.h, 21
gerror_to_string
    gerror.c, 55
graph.c
    graph_add_edge, 56
    graph_create, 57
    graph_destroy, 57
    graph_get_label_at, 57
    graph_set_label_at, 58
graph.h
    graph_add_edge, 22
    graph_create, 23
    graph_destroy, 23
    graph_get_label_at, 23
    graph_set_label_at, 25
    graph_t, 22
graph_add_edge
    graph.c, 56
    graph.h, 22
graph_create
    graph.c, 57
    graph.h, 23
graph_destroy
    graph.c, 57
    graph.h, 23
graph_get_label_at
    graph.c, 57
    graph.h, 23
graph_set_label_at
    graph.c, 58
    graph.h, 25
graph_t, 5
    adj, 6
    E, 6
    graph.h, 22
    label, 6
    member_size, 6
    V, 6

head
    queue_t, 10
    stack_t, 15

include/gerror.h, 19
include/graph.h, 21
include/priority_queue.h, 25
include/queue.h, 30
include/red_black_tree.h, 34
include/stack.h, 41
include/trie.h, 44
include/vector.h, 49

LEFT
    priority_queue.c, 59
label
    graph_t, 6
left
    redblacknode_t, 11
left_rotate
    red_black_tree.c, 68

max_heapify
priority_queue.c, 60
member_size
graph_t, 6
priority_queue_t, 7
queue_t, 10
redblacktree_t, 12
stack_t, 15
trie_t, 17
vector_t, 18

NBYTE
trie.h, 46
next
qnode_t, 9
snode_t, 14
node_at
trie.c, 79
nswap
priority_queue.c, 60

PARENT
priority_queue.c, 60
parent
redblacknode_t, 11
pqueue_add
priority_queue.c, 60
priority_queue.h, 28
pqueue_compare_function
priority_queue.h, 27
pqueue_create
priority_queue.c, 61
priority_queue.h, 28
pqueue_default_compare_function
priority_queue.c, 61
pqueue_destroy
priority_queue.c, 61
priority_queue.h, 28
pqueue_extract
priority_queue.c, 62
priority_queue.h, 29
pqueue_max_priority
priority_queue.c, 62
priority_queue.h, 29
pqueue_set_compare_function
priority_queue.c, 63
priority_queue.h, 30
pqueue_t
priority_queue.h, 27
prev
qnode_t, 9
snode_t, 14
priority_queue.c
LEFT, 59
max_heapify, 60
nswap, 60
PARENT, 60
pqueue_add, 60
pqueue_create, 61
pqueue_default_compare_function, 61
pqueue_destroy, 61
pqueue_extract, 62
pqueue_max_priority, 62
pqueue_set_compare_function, 63
RIGHT, 60
priority_queue.h
pqueue_add, 28
pqueue_compare_function, 27
pqueue_create, 28
pqueue_destroy, 28
pqueue_extract, 29
pqueue_max_priority, 29
pqueue_set_compare_function, 30
pqueue_t, 27
priority_queue_t, 27
queue_priority_t, 27
priority_queue_t, 7
compare, 7
compare_argument, 7
member_size, 7
priority_queue.h, 27
queue, 7
size, 8

qnode_t, 8
data, 8
next, 9
prev, 9
queue.h, 31
queue
priority_queue_t, 7
queue.c
queue_create, 64
queue_dequeue, 64
queue_destroy, 65
queue_enqueue, 65
queue_remove, 65
queue.h
qnode_t, 31
queue_create, 32
queue_dequeue, 32
queue_destroy, 32
queue_enqueue, 33
queue_remove, 33
queue_t, 31
queue_create
queue.c, 64
queue.h, 32
queue_dequeue
queue.c, 64
queue.h, 32
queue_destroy
queue.c, 65
queue.h, 32
queue_enqueue
queue.c, 65
queue.h, 33
queue_priority_t

priority_queue.h, 27
queue_remove
queue.c, 65
queue.h, 33
queue_t, 9
head, 10
member_size, 10
queue.h, 31
size, 10
tail, 10

RIGHT
priority_queue.c, 60
rbc_t
red_black_tree.c, 67
rbcolor_t
red_black_tree.h, 36
rbcomp_t
red_black_tree.h, 36
rbnode_destroy
red_black_tree.c, 68
rbnode_is_black
red_black_tree.c, 68
rbnode_is_red
red_black_tree.c, 68
rbnode_t
red_black_tree.h, 35
rbtree_add
red_black_tree.c, 68
red_black_tree.h, 36
rbtree_compare_function
red_black_tree.h, 35
rbtree_create
red_black_tree.c, 69
red_black_tree.h, 37
rbtree_create_double_black
red_black_tree.c, 69
rbtree_default_compare_function
red_black_tree.c, 69
rbtree_delete_fixup
red_black_tree.c, 69
rbtree_destroy
red_black_tree.c, 70
red_black_tree.h, 37
rbtree_find_minimal_node
red_black_tree.c, 70
rbtree_find_node
red_black_tree.c, 70
red_black_tree.h, 38
rbtree_identify_case
red_black_tree.c, 71
rbtree_insert_fixup
red_black_tree.c, 71
rbtree_max_node
red_black_tree.c, 71
red_black_tree.h, 38
rbtree_max_value
red_black_tree.c, 71
red_black_tree.h, 38
rbtree_min_node
red_black_tree.c, 73
red_black_tree.h, 39
rbtree_min_value
red_black_tree.c, 73
red_black_tree.h, 39
rbtree_remove_double_black
red_black_tree.c, 74
rbtree_remove_item
red_black_tree.c, 74
red_black_tree.h, 40
rbtree_remove_node
red_black_tree.c, 74
red_black_tree.h, 40
rbtree_set_compare_function
red_black_tree.c, 75
red_black_tree.h, 40
rbtree_t
red_black_tree.h, 35
rbtree_transplant
red_black_tree.c, 75
red_black_tree.c
create_node, 67
fix_insert_case, 67
left_rotate, 68
rbc_t, 67
rbnode_destroy, 68
rbnode_is_black, 68
rbnode_is_red, 68
rbtree_add, 68
rbtree_create, 69
rbtree_create_double_black, 69
rbtree_default_compare_function, 69
rbtree_delete_fixup, 69
rbtree_destroy, 70
rbtree_find_minimal_node, 70
rbtree_find_node, 70
rbtree_identify_case, 71
rbtree_insert_fixup, 71
rbtree_max_node, 71
rbtree_max_value, 71
rbtree_min_node, 73
rbtree_min_value, 73
rbtree_remove_double_black, 74
rbtree_remove_item, 74
rbtree_remove_node, 74
rbtree_set_compare_function, 75
rbtree_transplant, 75
right_rotate, 75
red_black_tree.h
rbcolor_t, 36
rbcomp_t, 36
rbnode_t, 35
rbtree_add, 36
rbtree_compare_function, 35
rbtree_create, 37
rbtree_destroy, 37
rbtree_find_node, 38

rbtree_max_node, 38
rbtree_max_value, 38
rbtree_min_node, 39
rbtree_min_value, 39
rbtree_remove_item, 40
rbtree_remove_node, 40
rbtree_set_compare_function, 40
rbtree_t, 35
redblacknode_t, 35
redblacktree_t, 36
redblacknode_t, 10
color, 11
data, 11
left, 11
parent, 11
red_black_tree.h, 35
right, 11
redblacktree_t, 12
compare, 12
compare_argument, 12
member_size, 12
red_black_tree.h, 36
root, 13
size, 13
right
redblacknode_t, 11
right_rotate
red_black_tree.c, 75
root
redblacktree_t, 13
trie_t, 17

size
priority_queue_t, 8
queue_t, 10
redblacktree_t, 13
stack_t, 15
trie_t, 17
vector_t, 18
snode_t, 13
data, 14
next, 14
prev, 14
stack.h, 42
src/gerror.c, 54
src/graph.c, 55
src/priority_queue.c, 58
src/queue.c, 63
src/red_black_tree.c, 66
src/stack.c, 76
src/trie.c, 78
src/vector.c, 81
stack.c
stack_create, 76
stack_destroy, 77
stack_pop, 77
stack_push, 77
stack.h
snode_t, 42
stack_create, 43
stack_destroy, 43
stack_pop, 43
stack_push, 44
stack_t, 42
stack_create
stack.c, 76
stack.h, 43
stack_destroy
stack.c, 77
stack.h, 43
stack_pop
stack.c, 77
stack.h, 43
stack_push
stack.c, 77
stack.h, 44
stack_t, 14
head, 15
member_size, 15
size, 15
stack.h, 42

tail
queue_t, 10
tnode_t, 15
children, 16
trie.h, 46
value, 16
trie.c
node_at, 79
trie_add_element, 79
trie_create, 79
trie_destroy, 79
trie_destroy_tnode, 80
trie_get_element, 80
trie_get_node_or_allocate, 80
trie_remove_element, 80
trie_set_element, 81
trie.h
NBYTE, 46
tnode_t, 46
trie_add_element, 46
trie_create, 47
trie_destroy, 47
trie_get_element, 47
trie_get_node_or_allocate, 48
trie_remove_element, 48
trie_set_element, 48
trie_t, 46
trie_add_element
trie.c, 79
trie.h, 46
trie_create
trie.c, 79
trie.h, 47
trie_destroy
trie.c, 79
trie.h, 47

trie_destroy_tnode
trie.c, 80
trie_get_element
trie.c, 80
trie.h, 47
trie_get_node_or_allocate
trie.c, 80
trie.h, 48
trie_remove_element
trie.c, 80
trie.h, 48
trie_set_element
trie.c, 81
trie.h, 48
trie_t, 16
member_size, 17
root, 17
size, 17
trie.h, 46

V
graph_t, 6
VECTOR_MIN_SIZ
vector.c, 82
value
tnode_t, 16
vector.c
VECTOR_MIN_SIZ, 82
vector_add, 83
vector_at, 83
vector_create, 83
vector_destroy, 84
vector_get_min_buf_siz, 84
vector_min_siz, 86
vector_ptr_at, 84
vector_resize_buffer, 85
vector_set_elem_at, 85
vector_set_min_buf_siz, 86
vector.h
vector_add, 51
vector_at, 51
vector_create, 51
vector_destroy, 52
vector_get_min_buf_siz, 52
vector_ptr_at, 52
vector_resize_buffer, 53
vector_set_elem_at, 53
vector_set_min_buf_siz, 54
vector_t, 50
vector_add
vector.c, 83
vector.h, 51
vector_at
vector.c, 83
vector.h, 51
vector_create
vector.c, 83
vector.h, 51
vector_destroy
vector.c, 84
vector.h, 52
vector_get_min_buf_siz
vector.c, 84
vector.h, 52
vector_min_siz
vector.c, 86
vector_ptr_at
vector.c, 84
vector.h, 52
vector_resize_buffer
vector.c, 85
vector.h, 53
vector_set_elem_at
vector.c, 85
vector.h, 53
vector_set_min_buf_siz
vector.c, 86
vector.h, 54
vector_t, 17
buffer_size, 18
data, 18
member_size, 18
size, 18
vector.h, 50