

libgenerics

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>Class Index</b>	<b>1</b>
1.1	Class List . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Class Documentation</b>	<b>5</b>
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	qnode_t Struct Reference . . . . .	6
3.2.1	Detailed Description . . . . .	6
3.2.2	Member Data Documentation . . . . .	7
3.2.2.1	data . . . . .	7
3.2.2.2	next . . . . .	7
3.2.2.3	prev . . . . .	7
3.3	queue_t Struct Reference . . . . .	7
3.3.1	Detailed Description . . . . .	7
3.3.2	Member Data Documentation . . . . .	8

3.3.2.1	head	8
3.3.2.2	member_size	8
3.3.2.3	size	8
3.3.2.4	tail	8
3.4	snode_t Struct Reference	8
3.4.1	Detailed Description	8
3.4.2	Member Data Documentation	8
3.4.2.1	data	8
3.4.2.2	next	8
3.4.2.3	prev	8
3.5	stack_t Struct Reference	9
3.5.1	Detailed Description	9
3.5.2	Member Data Documentation	9
3.5.2.1	head	9
3.5.2.2	member_size	9
3.5.2.3	size	9
3.6	tnode_t Struct Reference	10
3.6.1	Detailed Description	10
3.6.2	Member Data Documentation	10
3.6.2.1	children	10
3.6.2.2	value	10
3.7	trie_t Struct Reference	10
3.7.1	Detailed Description	11
3.7.2	Member Data Documentation	11
3.7.2.1	member_size	11
3.7.2.2	root	11
3.7.2.3	size	11
3.8	vector_t Struct Reference	11
3.8.1	Member Data Documentation	11
3.8.1.1	buffer_size	11
3.8.1.2	data	11
3.8.1.3	member_size	11
3.8.1.4	size	11

<b>4 File Documentation</b>	<b>13</b>
4.1 include/graph.h File Reference	13
4.1.1 Typedef Documentation	14
4.1.1.1 graph_t	14
4.1.2 Function Documentation	14
4.1.2.1 graph_add_edge(graph_t *g, size_t from, size_t to)	14
4.1.2.2 graph_create(graph_t *g, size_t size, size_t member_size)	15
4.1.2.3 graph_destroy(graph_t *g)	15
4.1.2.4 graph_get_label_at(graph_t *g, size_t index)	15
4.1.2.5 graph_set_label_at(graph_t *g, size_t index, void *label)	15
4.2 include/queue.h File Reference	16
4.2.1 Typedef Documentation	17
4.2.1.1 qnode_t	17
4.2.1.2 queue_t	17
4.2.2 Function Documentation	17
4.2.2.1 queue_create(struct queue_t *q, size_t member_size)	17
4.2.2.2 queue_dequeue(struct queue_t *q)	17
4.2.2.3 queue_destroy(struct queue_t *q)	17
4.2.2.4 queue_enqueue(struct queue_t *q, void *e)	18
4.2.2.5 queue_remove(struct queue_t *q, struct qnode_t *node)	18
4.3 include/stack.h File Reference	18
4.3.1 Typedef Documentation	19
4.3.1.1 snode_t	19
4.3.1.2 stack_t	19
4.3.2 Function Documentation	19
4.3.2.1 stack_create(struct stack_t *q, size_t member_size)	19
4.3.2.2 stack_destroy(struct stack_t *q)	20
4.3.2.3 stack_pop(struct stack_t *q)	20
4.3.2.4 stack_push(struct stack_t *q, void *e)	20
4.4 include/trie.h File Reference	20

4.4.1	Macro Definition Documentation . . . . .	22
4.4.1.1	NBYTE . . . . .	22
4.4.2	Typedef Documentation . . . . .	22
4.4.2.1	tnode_t . . . . .	22
4.4.2.2	trie_t . . . . .	22
4.4.3	Function Documentation . . . . .	22
4.4.3.1	trie_add_element(struct trie_t *t, void *string, size_t size, void *elem) . . . . .	22
4.4.3.2	trie_create(struct trie_t *t, size_t member_size) . . . . .	22
4.4.3.3	trie_destroy(struct trie_t *t) . . . . .	22
4.4.3.4	trie_get_element(struct trie_t *t, void *string, size_t size) . . . . .	23
4.4.3.5	trie_remove_element(struct trie_t *t, void *string, size_t size) . . . . .	23
4.4.3.6	trie_set_element(struct trie_t *t, void *string, size_t size, void *elem) . . . . .	23
4.5	include/vector.h File Reference . . . . .	23
4.5.1	Typedef Documentation . . . . .	25
4.5.1.1	vector_t . . . . .	25
4.5.2	Function Documentation . . . . .	25
4.5.2.1	vector_add(vector_t *v, void *elem) . . . . .	25
4.5.2.2	vector_at(vector_t *v, size_t index) . . . . .	25
4.5.2.3	vector_create(vector_t *v, size_t initial_size, size_t member_size) . . . . .	25
4.5.2.4	vector_destroy(vector_t *v) . . . . .	25
4.5.2.5	vector_get_min_buf_siz(void) . . . . .	26
4.5.2.6	vector_resize_buffer(vector_t *v, size_t new_size) . . . . .	26
4.5.2.7	vector_set_elem_at(vector_t *v, size_t index, void *elem) . . . . .	26
4.5.2.8	vector_set_min_buf_siz(size_t new_min_buf_size) . . . . .	26
4.6	src/graph.c File Reference . . . . .	27
4.6.1	Function Documentation . . . . .	27
4.6.1.1	graph_add_edge(graph_t *g, size_t from, size_t to) . . . . .	27
4.6.1.2	graph_create(graph_t *g, size_t size, size_t member_size) . . . . .	28
4.6.1.3	graph_destroy(graph_t *g) . . . . .	28
4.6.1.4	graph_get_label_at(graph_t *g, size_t index) . . . . .	28

4.6.1.5	graph_set_label_at(graph_t *g, size_t index, void *label)	28
4.7	src/queue.c File Reference	29
4.7.1	Function Documentation	29
4.7.1.1	queue_create(struct queue_t *q, size_t member_size)	29
4.7.1.2	queue_dequeue(struct queue_t *q)	29
4.7.1.3	queue_destroy(struct queue_t *q)	30
4.7.1.4	queue_enqueue(struct queue_t *q, void *e)	30
4.7.1.5	queue_remove(struct queue_t *q, struct qnode_t *node)	30
4.8	src/stack.c File Reference	30
4.8.1	Function Documentation	31
4.8.1.1	stack_create(struct stack_t *s, size_t member_size)	31
4.8.1.2	stack_destroy(struct stack_t *s)	31
4.8.1.3	stack_pop(struct stack_t *s)	32
4.8.1.4	stack_push(struct stack_t *s, void *e)	32
4.9	src/trie.c File Reference	32
4.9.1	Function Documentation	33
4.9.1.1	node_at(struct trie_t *t, void *string, size_t size)	33
4.9.1.2	node_at_and_allocate(struct trie_t *t, void *string, size_t size)	33
4.9.1.3	trie_add_element(struct trie_t *t, void *string, size_t size, void *elem)	33
4.9.1.4	trie_create(struct trie_t *t, size_t member_size)	33
4.9.1.5	trie_destroy(struct trie_t *t)	33
4.9.1.6	trie_destroy_tnode(struct tnode_t *node)	34
4.9.1.7	trie_get_element(struct trie_t *t, void *string, size_t size)	34
4.9.1.8	trie_remove_element(struct trie_t *t, void *string, size_t size)	34
4.9.1.9	trie_set_element(struct trie_t *t, void *string, size_t size, void *elem)	34
4.10	src/vector.c File Reference	35
4.10.1	Macro Definition Documentation	35
4.10.1.1	VECTOR_MIN_SIZ	35
4.10.2	Function Documentation	35
4.10.2.1	vector_add(vector_t *v, void *elem)	35
4.10.2.2	vector_at(vector_t *v, size_t index)	36
4.10.2.3	vector_create(vector_t *v, size_t initial_buf_siz, size_t member_size)	36
4.10.2.4	vector_destroy(vector_t *v)	36
4.10.2.5	vector_get_min_buf_siz(void)	36
4.10.2.6	vector_resize_buffer(vector_t *v, size_t n_elements)	37
4.10.2.7	vector_set_elem_at(vector_t *v, size_t index, void *elem)	37
4.10.2.8	vector_set_min_buf_siz(size_t new_min_buf_siz)	37
4.10.3	Variable Documentation	37
4.10.3.1	vector_min_siz	37





# Chapter 1

## Class Index

### 1.1 Class List

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

<a href="#">graph_t</a>	5
<a href="#">qnode_t</a>	6
<a href="#">queue_t</a>	7
<a href="#">snode_t</a>	8
<a href="#">stack_t</a>	9
<a href="#">tnode_t</a>	10
<a href="#">trie_t</a>	10
<a href="#">vector_t</a>	11



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">include/graph.h</a>	13
<a href="#">include/queue.h</a>	16
<a href="#">include/stack.h</a>	18
<a href="#">include/trie.h</a>	20
<a href="#">include/vector.h</a>	23
<a href="#">src/graph.c</a>	27
<a href="#">src/queue.c</a>	29
<a href="#">src/stack.c</a>	30
<a href="#">src/trie.c</a>	32
<a href="#">src/vector.c</a>	35



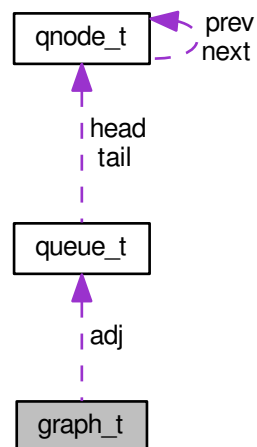
## Chapter 3

# 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 `struct queue_t* graph_t::adj`

3.1.2.2 `size_t graph_t::E`

3.1.2.3 `void* graph_t::label`

3.1.2.4 `size_t graph_t::member_size`

3.1.2.5 `size_t graph_t::V`

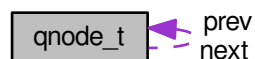
The documentation for this struct was generated from the following file:

- [include/graph.h](#)

## 3.2 qnode\_t Struct Reference

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

Collaboration diagram for `qnode_t`:



### Public Attributes

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

### 3.2.1 Detailed Description

queue node.

### 3.2.2 Member Data Documentation

3.2.2.1 void\* qnode\_t::data

3.2.2.2 struct qnode\_t\* qnode\_t::next

3.2.2.3 struct qnode\_t\* qnode\_t::prev

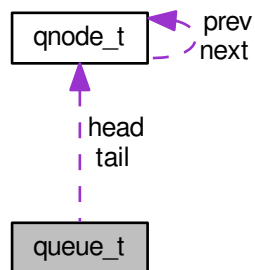
The documentation for this struct was generated from the following file:

- [include/queue.h](#)

## 3.3 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](#)

### 3.3.1 Detailed Description

Represents a queue structure.

### 3.3.2 Member Data Documentation

3.3.2.1 `struct qnode_t* queue_t::head`

3.3.2.2 `size_t queue_t::member_size`

3.3.2.3 `size_t queue_t::size`

3.3.2.4 `struct qnode_t* queue_t::tail`

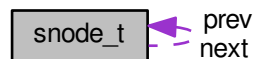
The documentation for this struct was generated from the following file:

- [include/queue.h](#)

## 3.4 snode\_t Struct Reference

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

Collaboration diagram for `snode_t`:



### Public Attributes

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

### 3.4.1 Detailed Description

node of a stack

### 3.4.2 Member Data Documentation

3.4.2.1 `void* snode_t::data`

3.4.2.2 `struct snode_t* snode_t::next`

3.4.2.3 `struct snode_t* snode_t::prev`

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

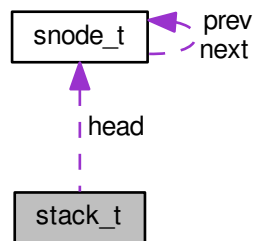
- [include/stack.h](#)



## 3.5 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](#)

### 3.5.1 Detailed Description

represents the stack structure.

### 3.5.2 Member Data Documentation

3.5.2.1 `struct snode_t* stack_t::head`

3.5.2.2 `size_t stack_t::member_size`

3.5.2.3 `size_t stack_t::size`

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

- `include/`[stack.h](#)

### 3.6 tnode\_t Struct Reference

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

Collaboration diagram for tnode\_t:



#### Public Attributes

- void \* [value](#)
- struct [tnode\\_t](#) \* [children](#) [NBYTE]

#### 3.6.1 Detailed Description

node of a [trie\\_t](#) element.

#### 3.6.2 Member Data Documentation

3.6.2.1 struct [tnode\\_t](#)\* [tnode\\_t::children](#)[NBYTE]

3.6.2.2 void\* [tnode\\_t::value](#)

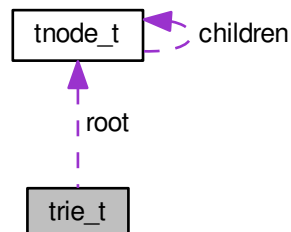
The documentation for this struct was generated from the following file:

- [include/trie.h](#)

### 3.7 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](#)

### 3.7.1 Detailed Description

Represents the trie structure.

### 3.7.2 Member Data Documentation

3.7.2.1 `size_t` `trie_t::member_size`

3.7.2.2 `struct tnode_t` `trie_t::root`

3.7.2.3 `size_t` `trie_t::size`

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

- `include/trie.h`

## 3.8 vector\_t Struct Reference

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

## Public Attributes

- `void *` [data](#)
- `size_t` [size](#)
- `size_t` [buffer\\_size](#)
- `size_t` [member\\_size](#)

### 3.8.1 Member Data Documentation

3.8.1.1 `size_t` `vector_t::buffer_size`

3.8.1.2 `void*` `vector_t::data`

3.8.1.3 `size_t` `vector_t::member_size`

3.8.1.4 `size_t` `vector_t::size`

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

- `include/vector.h`



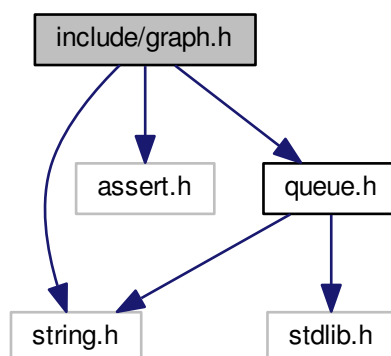
## Chapter 4

# File Documentation

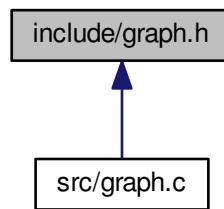
### 4.1 include/graph.h File Reference

```
#include <string.h>
#include <assert.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

- void [graph\\_create](#) ([graph\\_t](#) \*g, size\_t size, size\_t member\_size)
- void [graph\\_add\\_edge](#) ([graph\\_t](#) \*g, size\_t from, size\_t to)
- void \* [graph\\_get\\_label\\_at](#) ([graph\\_t](#) \*g, size\_t index)
- void [graph\\_set\\_label\\_at](#) ([graph\\_t](#) \*g, size\_t index, void \*label)
- void [graph\\_destroy](#) ([graph\\_t](#) \*g)

### 4.1.1 Typedef Documentation

#### 4.1.1.1 typedef struct [graph\\_t](#) [graph\\_t](#)

Graph structure and elements.

### 4.1.2 Function Documentation

#### 4.1.2.1 void [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

<i>g</i>	pointer to a graph structure;
<i>from</i>	index of the first vertex;
<i>to</i>	index of the incident vertex.

4.1.2.2 void 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

<i>g</i>	pointer to a graph structure;
<i>member_size</i>	size of the elements that will be indexed by <i>g</i>

4.1.2.3 void graph\_destroy ( graph\_t \* *g* )

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

## Parameters

<i>g</i>	pointer to a graph structure;
----------	-------------------------------

4.1.2.4 void\* graph\_get\_label\_at ( graph\_t \* *g*, size\_t *index* )

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

## Parameters

<i>g</i>	pointer to a graph structure;
<i>index</i>	index of the vertex;

## Returns

pointer to the label of the vertex positioned in *index*.

4.1.2.5 void graph\_set\_label\_at ( graph\_t \* *g*, size\_t *index*, void \* *label* )

Sets the label at the *index* to *label*.

## Parameters

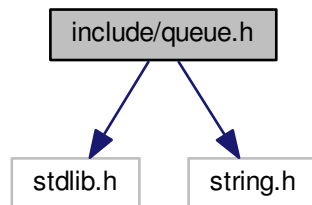
<i>g</i>	pointer to a graph structure;
<i>index</i>	index of the vertex;
<i>label</i>	the new label of the vertex positioned in <i>index</i>

## 4.2 include/queue.h File Reference

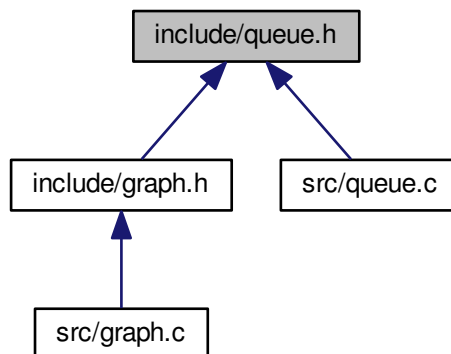
```
#include <stdlib.h>
```

```
#include <string.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

- void [queue\\_create](#) (struct [queue\\_t](#) \*q, size\_t member\_size)
- void [queue\\_enqueue](#) (struct [queue\\_t](#) \*q, void \*e)
- void \* [queue\\_dequeue](#) (struct [queue\\_t](#) \*q)
- void [queue\\_destroy](#) (struct [queue\\_t](#) \*q)
- void \* [queue\\_remove](#) (struct [queue\\_t](#) \*q, struct [qnode\\_t](#) \*node)

### 4.2.1 Typedef Documentation

#### 4.2.1.1 typedef struct qnode\_t qnode\_t

queue node.

#### 4.2.1.2 typedef struct queue\_t queue\_t

Represents a queue structure.

### 4.2.2 Function Documentation

#### 4.2.2.1 void queue\_create ( struct queue\_t \* q, size\_t member\_size )

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

##### Parameters

<i>q</i>	pointer to a queue structure;
<i>member_size</i>	size of the elements that will be indexed by q

#### 4.2.2.2 void\* queue\_dequeue ( struct queue\_t \* q )

Dequeues the first element of the queue q

##### Parameters

<i>q</i>	pointer to a queue structure;
----------	-------------------------------

##### Returns

a pointer to the element that must be freed;

#### 4.2.2.3 void queue\_destroy ( struct queue\_t \* q )

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

## Parameters

<i>q</i>	pointer to a queue structure;
----------	-------------------------------

4.2.2.4 void queue\_enqueue ( struct queue\_t \* *q*, void \* *e* )

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

## Parameters

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

4.2.2.5 void\* queue\_remove ( struct queue\_t \* *q*, struct qnode\_t \* *node* )

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

## Parameters

<i>q</i>	pointer to a queue structure;
<i>node</i>	element to be removed from the queue

## Returns

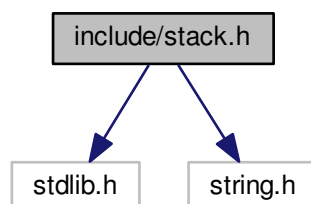
a pointer to the value of the node just removed

## 4.3 include/stack.h File Reference

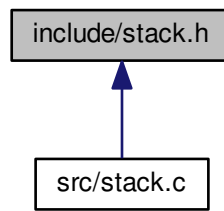
```
#include <stdlib.h>
```

```
#include <string.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

- void [stack\\_create](#) (struct [stack\\_t](#) \*q, size\_t member\_size)
- void [stack\\_push](#) (struct [stack\\_t](#) \*q, void \*e)
- void \* [stack\\_pop](#) (struct [stack\\_t](#) \*q)
- void [stack\\_destroy](#) (struct [stack\\_t](#) \*q)

### 4.3.1 Typedef Documentation

#### 4.3.1.1 typedef struct [snode\\_t](#) [snode\\_t](#)

node of a stack

#### 4.3.1.2 typedef struct [stack\\_t](#) [stack\\_t](#)

represents the stack structure.

### 4.3.2 Function Documentation

#### 4.3.2.1 void [stack\\_create](#) ( struct [stack\\_t](#) \* s, size\_t member\_size )

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

## Parameters

<i>s</i>	pointer to a stack structure;
<i>member_size</i>	size of the elements that will be indexed by <i>s</i>

## 4.3.2.2 void stack\_destroy ( struct stack\_t \* s )

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

## Parameters

<i>s</i>	pointer to a stack structure;
----------	-------------------------------

## 4.3.2.3 void\* stack\_pop ( struct stack\_t \* s )

Pops the first element of the stack *s*.

## Parameters

<i>s</i>	pointer to a stack structure;
----------	-------------------------------

## Returns

a pointer to the element that must be freed;

## 4.3.2.4 void stack\_push ( struct stack\_t \* s, void \* e )

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

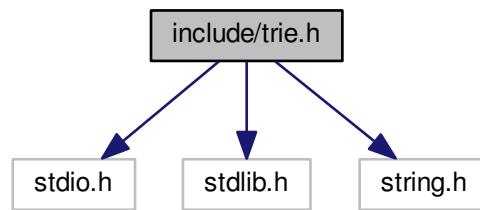
## Parameters

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

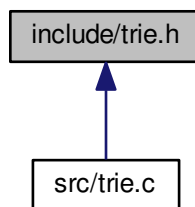
## 4.4 include/trie.h File Reference

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

- void [trie\\_create](#) (struct [trie\\_t](#) \*t, size\_t member\_size)
- void [trie\\_destroy](#) (struct [trie\\_t](#) \*t)
- void [trie\\_add\\_element](#) (struct [trie\\_t](#) \*t, void \*string, size\_t size, void \*elem)
- void \* [trie\\_remove\\_element](#) (struct [trie\\_t](#) \*t, void \*string, size\_t size)
- void \* [trie\\_get\\_element](#) (struct [trie\\_t](#) \*t, void \*string, size\_t size)
- void [trie\\_set\\_element](#) (struct [trie\\_t](#) \*t, void \*string, size\_t size, void \*elem)

#### 4.4.1 Macro Definition Documentation

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

#### 4.4.2 Typedef Documentation

##### 4.4.2.1 `typedef struct tnode_t tnode_t`

node of a [trie\\_t](#) element.

##### 4.4.2.2 `typedef struct trie_t trie_t`

Represents the trie structure.

#### 4.4.3 Function Documentation

##### 4.4.3.1 `void 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

<i>t</i>	pointer to the trie structure;
<i>string</i>	pointer to the string of bytes to map <code>elem</code> ;
<i>size</i>	size of the string of bytes
<i>elem</i>	pointer to the element to add

##### 4.4.3.2 `void trie_create ( struct trie_t * t, size_t member_size )`

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

###### Parameters

<i>t</i>	pointer to the allocated struct <a href="#">trie_t</a> ;
<i>member_size</i>	size in bytes of the indexed elements by the trie.

##### 4.4.3.3 `void trie_destroy ( struct trie_t * t )`

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

###### Parameters

<i>t</i>	pointer to the structure
----------	--------------------------

#### 4.4.3.4 void\* trie\_get\_element ( struct trie\_t \* t, void \* string, size\_t size )

Returns the element mapped by *string*.

##### Parameters

<i>t</i>	pointer to the structure;
<i>string</i>	pointer to the string of bytes to map elem;
<i>size</i>	size of the string of bytes.

##### Returns

The removed element mapped by *string*.

#### 4.4.3.5 void\* trie\_remove\_element ( struct trie\_t \* t, void \* string, size\_t size )

Removes the element mapped by *string*.

##### Parameters

<i>t</i>	pointer to the structure <a href="#">trie_t</a> ;
<i>string</i>	pointer to the string of bytes to map elem;
<i>size</i>	size of the string of bytes.

##### Returns

pointer to the removed element

#### 4.4.3.6 void 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.

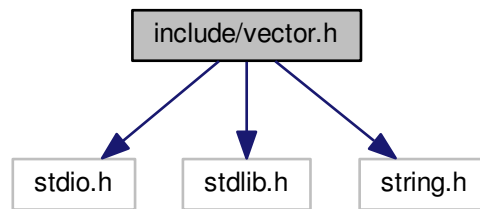
##### Parameters

<i>t</i>	pointer to the structure;
<i>string</i>	pointer to the string of bytes to map elem;
<i>size</i>	size of the string of bytes.
<i>elem</i>	pointer to the element to add

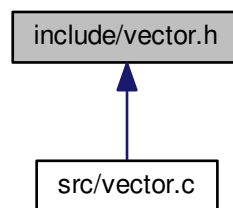
## 4.5 include/vector.h File Reference

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

- void [vector\\_create](#) ([vector\\_t](#) \*v, size\_t initial\_size, size\_t member\_size)
- void [vector\\_destroy](#) ([vector\\_t](#) \*v)
- void [vector\\_resize\\_buffer](#) ([vector\\_t](#) \*v, size\_t new\_size)
- void \* [vector\\_at](#) ([vector\\_t](#) \*v, size\_t index)
- void [vector\\_set\\_elem\\_at](#) ([vector\\_t](#) \*v, size\_t index, void \*elem)
- void [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)



## 4.5.1 Typedef Documentation

### 4.5.1.1 typedef struct vector\_t vector\_t

## 4.5.2 Function Documentation

### 4.5.2.1 void vector\_add ( vector\_t \* v, void \* elem )

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

#### Parameters

<code>v</code>	a pointer to <code>vector_t</code>
<code>elem</code>	the element to be add in <code>v</code>

### 4.5.2.2 void\* vector\_at ( vector\_t \* v, size\_t index )

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

#### Parameters

<code>v</code>	a pointer to <code>vector_t</code>
<code>index</code>	index of the position

#### Returns

a pointer to the member at `index`

### 4.5.2.3 void 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

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

### 4.5.2.4 void vector\_destroy ( vector\_t \* v )

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

## Parameters

<i>v</i>	a pointer to <code>vector_t</code> structure
----------	--

4.5.2.5 `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`

4.5.2.6 `void vector_resize_buffer ( vector_t * v, size_t n_elements )`

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

## Parameters

<i>v</i>	a pointer to <code>vector_t</code> structure.
<i>new_size</i>	the new size of the <code>v</code>

4.5.2.7 `void 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

<i>v</i>	a pointer to <code>vector_t</code>
<i>index</i>	index of the position
<i>elem</i>	the element to be set in <code>v</code>

4.5.2.8 `void vector_set_min_buf_siz ( size_t new_min_buf_siz )`

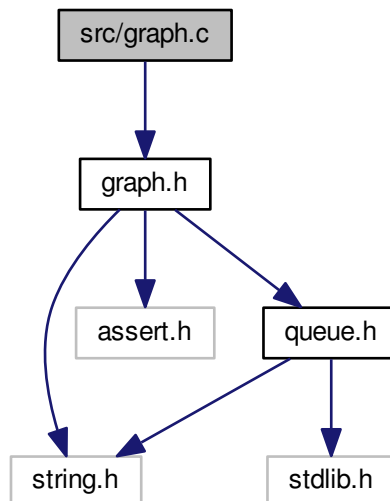
Set the `vector_min_siz`: a private variable that holds the minimal number of elements that `vector_t` will index. This variable is important for avoid multiple small resizes in the `vector_t` container.

## Parameters

<i>new_min_buf_siz</i>	the new size of <code>vector_min_siz</code>
------------------------	---

## 4.6 src/graph.c File Reference

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



### Functions

- void [graph\\_create](#) ([graph\\_t](#) \*g, [size\\_t](#) size, [size\\_t](#) member\_size)
- void [graph\\_add\\_edge](#) ([graph\\_t](#) \*g, [size\\_t](#) from, [size\\_t](#) to)
- void \* [graph\\_get\\_label\\_at](#) ([graph\\_t](#) \*g, [size\\_t](#) index)
- void [graph\\_set\\_label\\_at](#) ([graph\\_t](#) \*g, [size\\_t](#) index, void \*label)
- void [graph\\_destroy](#) ([graph\\_t](#) \*g)

### 4.6.1 Function Documentation

#### 4.6.1.1 void [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

<i>g</i>	pointer to a graph structure;
<i>from</i>	index of the first vertex;
<i>to</i>	index of the incident vertex.

#### 4.6.1.2 void 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

<i>g</i>	pointer to a graph structure;
<i>member_size</i>	size of the elements that will be indexed by <i>g</i>

#### 4.6.1.3 void graph\_destroy ( graph\_t \* *g* )

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

##### Parameters

<i>g</i>	pointer to a graph structure;
----------	-------------------------------

#### 4.6.1.4 void\* graph\_get\_label\_at ( graph\_t \* *g*, size\_t *index* )

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

##### Parameters

<i>g</i>	pointer to a graph structure;
<i>index</i>	index of the vertex;

##### Returns

pointer to the label of the vertex positioned in *index*.

#### 4.6.1.5 void graph\_set\_label\_at ( graph\_t \* *g*, size\_t *index*, void \* *label* )

Sets the label at the *index* to *label*.

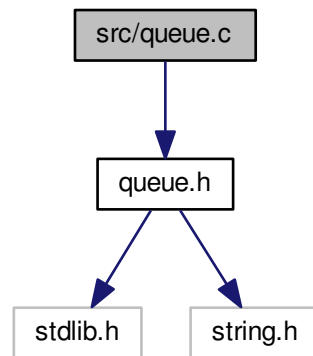
##### Parameters

<i>g</i>	pointer to a graph structure;
<i>index</i>	index of the vertex;
<i>label</i>	the new label of the vertex positioned in <i>index</i>

## 4.7 src/queue.c File Reference

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

Include dependency graph for queue.c:



### Functions

- void [queue\\_create](#) (struct [queue\\_t](#) \*q, size\_t member\_size)
- void [queue\\_enqueue](#) (struct [queue\\_t](#) \*q, void \*e)
- void \* [queue\\_dequeue](#) (struct [queue\\_t](#) \*q)
- void \* [queue\\_remove](#) (struct [queue\\_t](#) \*q, struct [qnode\\_t](#) \*node)
- void [queue\\_destroy](#) (struct [queue\\_t](#) \*q)

#### 4.7.1 Function Documentation

##### 4.7.1.1 void queue\_create ( struct queue\_t \* q, size\_t member\_size )

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

##### Parameters

<i>q</i>	pointer to a queue structure;
<i>member_size</i>	size of the elements that will be indexed by <code>q</code>

##### 4.7.1.2 void\* queue\_dequeue ( struct queue\_t \* q )

Dequeues the first element of the queue `q`

**Parameters**

<i>q</i>	pointer to a queue structure;
----------	-------------------------------

**Returns**

a pointer to the element that must be freed;

**4.7.1.3 void queue\_destroy ( struct queue\_t \* *q* )**

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

**Parameters**

<i>q</i>	pointer to a queue structure;
----------	-------------------------------

**4.7.1.4 void queue\_enqueue ( struct queue\_t \* *q*, void \* *e* )**

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

**Parameters**

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

**4.7.1.5 void\* queue\_remove ( struct queue\_t \* *q*, struct qnode\_t \* *node* )**

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

**Parameters**

<i>q</i>	pointer to a queue structure;
<i>node</i>	element to be removed from the queue

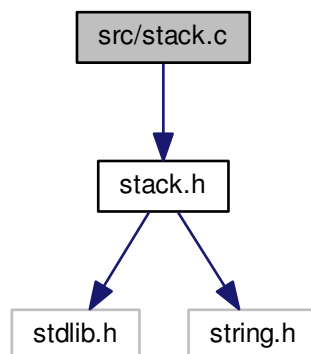
**Returns**

a pointer to the value of the node just removed

## 4.8 src/stack.c File Reference

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

Include dependency graph for stack.c:



## Functions

- void [stack\\_create](#) (struct [stack\\_t](#) \*s, size\_t member\_size)
- void [stack\\_push](#) (struct [stack\\_t](#) \*s, void \*e)
- void \* [stack\\_pop](#) (struct [stack\\_t](#) \*s)
- void [stack\\_destroy](#) (struct [stack\\_t](#) \*s)

### 4.8.1 Function Documentation

#### 4.8.1.1 void stack\_create ( struct stack\_t \* s, size\_t member\_size )

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

##### Parameters

<i>s</i>	pointer to a stack structure;
<i>member_size</i>	size of the elements that will be indexed by <i>s</i>

#### 4.8.1.2 void stack\_destroy ( struct stack\_t \* s )

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

##### Parameters

<i>s</i>	pointer to a stack structure;
----------	-------------------------------

#### 4.8.1.3 void\* stack\_pop ( struct stack\_t \* s )

Pops the first element of the stack *s*.

##### Parameters

<i>s</i>	pointer to a stack structure;
----------	-------------------------------

##### Returns

a pointer to the element that must be freed;

#### 4.8.1.4 void stack\_push ( struct stack\_t \* s, void \* e )

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

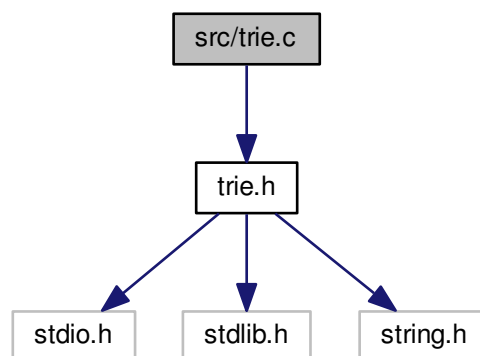
##### Parameters

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

## 4.9 src/trie.c File Reference

```
#include "trie.h"
```

Include dependency graph for trie.c:



## Functions

- [tnode\\_t \\* node\\_at\\_and\\_allocate](#) (struct [trie\\_t](#) \*t, void \*string, size\_t size)



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

## 4.9.1 Function Documentation

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

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

4.9.1.3 `void 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

<i>t</i>	pointer to the trie structure;
<i>string</i>	pointer to the string of bytes to map <code>elem</code> ;
<i>size</i>	size of the string of bytes
<i>elem</i>	pointer to the element to add

4.9.1.4 `void trie_create ( struct trie_t * t, size_t member_size )`

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

### Parameters

<i>t</i>	pointer to the allocated struct <code>trie_t</code> ;
<i>member_size</i>	size in bytes of the indexed elements by the trie.

4.9.1.5 `void trie_destroy ( struct trie_t * t )`

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

### Parameters

<i>t</i>	pointer to the structure
----------	--------------------------

4.9.1.6 void trie\_destroy\_tnode ( struct tnode\_t \* node )

4.9.1.7 void\* trie\_get\_element ( struct trie\_t \* t, void \* string, size\_t size )

Returns the element mapped by `string`.

#### Parameters

<i>t</i>	pointer to the structure;
<i>string</i>	pointer to the string of bytes to map elem;
<i>size</i>	size of the string of bytes.

#### Returns

The removed element mapped by `string`.

4.9.1.8 void\* trie\_remove\_element ( struct trie\_t \* t, void \* string, size\_t size )

Removes the element mapped by `string`.

#### Parameters

<i>t</i>	pointer to the structure <a href="#">trie_t</a> ;
<i>string</i>	pointer to the string of bytes to map elem;
<i>size</i>	size of the string of bytes.

#### Returns

pointer to the removed element

4.9.1.9 void 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.

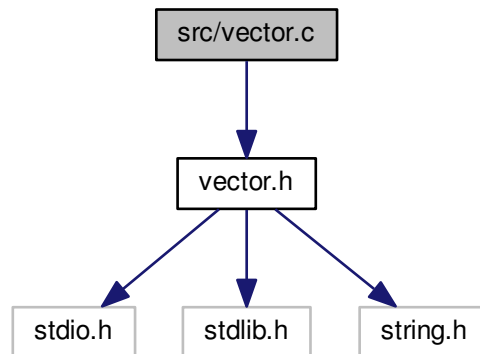
#### Parameters

<i>t</i>	pointer to the structure;
<i>string</i>	pointer to the string of bytes to map elem;
<i>size</i>	size of the string of bytes.
<i>elem</i>	pointer to the element to add

## 4.10 src/vector.c File Reference

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

Include dependency graph for vector.c:



### Macros

- `#define VECTOR_MIN_SIZ 8`

### Functions

- void `vector_create` (`vector_t` \*v, size\_t initial\_buf\_siz, size\_t member\_size)
- void `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)
- void `vector_resize_buffer` (`vector_t` \*v, size\_t n\_elements)
- void \* `vector_at` (`vector_t` \*v, size\_t index)
- void `vector_set_elem_at` (`vector_t` \*v, size\_t index, void \*elem)
- void `vector_add` (`vector_t` \*v, void \*elem)

### Variables

- size\_t `vector_min_siz` = `VECTOR_MIN_SIZ`

#### 4.10.1 Macro Definition Documentation

##### 4.10.1.1 `#define VECTOR_MIN_SIZ 8`

#### 4.10.2 Function Documentation

##### 4.10.2.1 void `vector_add` ( `vector_t` \* v, void \* elem )

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

## Parameters

<i>v</i>	a pointer to <a href="#">vector_t</a>
<i>elem</i>	the element to be add in <i>v</i>

## 4.10.2.2 void\* vector\_at ( vector\_t \* v, size\_t index )

Get the element in the *index* position indexed by the [vector\\_t](#) structure pointed by *v*.

## Parameters

<i>v</i>	a pointer to <a href="#">vector_t</a>
<i>index</i>	index of the position

## Returns

a pointer to the member at *index*

## 4.10.2.3 void vector\_create ( vector\_t \* v, size\_t initial\_buf\_siz, size\_t member\_size )

Populate the [vector\\_t](#) structure pointed by *v* and allocates *member\_size\*initial\_size* for initial buffer↵  
\_size.

## Parameters

<i>v</i>	a pointer to <a href="#">vector_t</a> structure already allocated;
<i>inicial_buf_size</i>	number of the members of the initial allocated buffer;
<i>member_size</i>	size of every member indexed by <i>v</i> .

## 4.10.2.4 void vector\_destroy ( vector\_t \* v )

Destroy the structure [vector\\_t](#) pointed by *v*.

## Parameters

<i>v</i>	a pointer to <a href="#">vector_t</a> structure
----------	---

## 4.10.2.5 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

## 4.10.2.6 void vector\_resize\_buffer ( vector\_t \* v, size\_t n\_elements )

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

## Parameters

<code>v</code>	a pointer to <code>vector_t</code> structure.
<code>new_size</code>	the new size of the <code>v</code>

## 4.10.2.7 void 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

<code>v</code>	a pointer to <code>vector_t</code>
<code>index</code>	index of the position
<code>elem</code>	the element to be set in <code>v</code>

## 4.10.2.8 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

<code>new_min_buf_siz</code>	the new size of <code>vector_min_siz</code>
------------------------------	---

## 4.10.3 Variable Documentation

## 4.10.3.1 size\_t vector\_min\_siz = VECTOR\_MIN\_SIZ



# Index

adj  
graph\_t, 6

buffer\_size  
vector\_t, 11

children  
tnode\_t, 10

data  
qnode\_t, 7  
snode\_t, 8  
vector\_t, 11

E  
graph\_t, 6

graph.c  
graph\_add\_edge, 27  
graph\_create, 27  
graph\_destroy, 28  
graph\_get\_label\_at, 28  
graph\_set\_label\_at, 28

graph.h  
graph\_add\_edge, 14  
graph\_create, 15  
graph\_destroy, 15  
graph\_get\_label\_at, 15  
graph\_set\_label\_at, 15  
graph\_t, 14

graph\_add\_edge  
graph.c, 27  
graph.h, 14

graph\_create  
graph.c, 27  
graph.h, 15

graph\_destroy  
graph.c, 28  
graph.h, 15

graph\_get\_label\_at  
graph.c, 28  
graph.h, 15

graph\_set\_label\_at  
graph.c, 28  
graph.h, 15

graph\_t, 5  
adj, 6  
E, 6  
graph.h, 14  
label, 6  
member\_size, 6

V, 6

head  
queue\_t, 8  
stack\_t, 9

include/graph.h, 13  
include/queue.h, 16  
include/stack.h, 18  
include/trie.h, 20  
include/vector.h, 23

label  
graph\_t, 6

member\_size  
graph\_t, 6  
queue\_t, 8  
stack\_t, 9  
trie\_t, 11  
vector\_t, 11

NBYTE  
trie.h, 22

next  
qnode\_t, 7  
snode\_t, 8

node\_at  
trie.c, 33  
node\_at\_and\_allocate  
trie.c, 33

prev  
qnode\_t, 7  
snode\_t, 8

qnode\_t, 6  
data, 7  
next, 7  
prev, 7  
queue.h, 17

queue.c  
queue\_create, 29  
queue\_dequeue, 29  
queue\_destroy, 30  
queue\_enqueue, 30  
queue\_remove, 30

queue.h  
qnode\_t, 17  
queue\_create, 17  
queue\_dequeue, 17

- queue\_destroy, 17
  - queue\_enqueue, 18
  - queue\_remove, 18
  - queue\_t, 17
- queue\_create
  - queue.c, 29
  - queue.h, 17
- queue\_dequeue
  - queue.c, 29
  - queue.h, 17
- queue\_destroy
  - queue.c, 30
  - queue.h, 17
- queue\_enqueue
  - queue.c, 30
  - queue.h, 18
- queue\_remove
  - queue.c, 30
  - queue.h, 18
- queue\_t, 7
  - head, 8
  - member\_size, 8
  - queue.h, 17
  - size, 8
  - tail, 8
- root
  - trie\_t, 11
- size
  - queue\_t, 8
  - stack\_t, 9
  - trie\_t, 11
  - vector\_t, 11
- snode\_t, 8
  - data, 8
  - next, 8
  - prev, 8
  - stack.h, 19
- src/graph.c, 27
- src/queue.c, 29
- src/stack.c, 30
- src/trie.c, 32
- src/vector.c, 35
- stack.c
  - stack\_create, 31
  - stack\_destroy, 31
  - stack\_pop, 31
  - stack\_push, 32
- stack.h
  - snode\_t, 19
  - stack\_create, 19
  - stack\_destroy, 20
  - stack\_pop, 20
  - stack\_push, 20
  - stack\_t, 19
- stack\_create
  - stack.c, 31
  - stack.h, 19
- stack\_destroy
  - stack.c, 31
  - stack.h, 20
- stack\_pop
  - stack.c, 31
  - stack.h, 20
- stack\_push
  - stack.c, 32
  - stack.h, 20
- stack\_t, 9
  - head, 9
  - member\_size, 9
  - size, 9
  - stack.h, 19
- tail
  - queue\_t, 8
- tnode\_t, 10
  - children, 10
  - trie.h, 22
  - value, 10
- trie.c
  - node\_at, 33
  - node\_at\_and\_allocate, 33
  - trie\_add\_element, 33
  - trie\_create, 33
  - trie\_destroy, 33
  - trie\_destroy\_tnode, 33
  - trie\_get\_element, 34
  - trie\_remove\_element, 34
  - trie\_set\_element, 34
- trie.h
  - NBYTE, 22
  - tnode\_t, 22
  - trie\_add\_element, 22
  - trie\_create, 22
  - trie\_destroy, 22
  - trie\_get\_element, 23
  - trie\_remove\_element, 23
  - trie\_set\_element, 23
  - trie\_t, 22
- trie\_add\_element
  - trie.c, 33
  - trie.h, 22
- trie\_create
  - trie.c, 33
  - trie.h, 22
- trie\_destroy
  - trie.c, 33
  - trie.h, 22
- trie\_destroy\_tnode
  - trie.c, 33
- trie\_get\_element
  - trie.c, 34
  - trie.h, 23
- trie\_remove\_element
  - trie.c, 34
  - trie.h, 23
- trie\_set\_element
  - trie.c, 34
  - trie.h, 23



- trie.c, [34](#)
- trie.h, [23](#)
- trie\_t, [10](#)
  - member\_size, [11](#)
  - root, [11](#)
  - size, [11](#)
  - trie.h, [22](#)
- V
  - graph\_t, [6](#)
- VECTOR\_MIN\_SIZ
  - vector.c, [35](#)
- value
  - tnode\_t, [10](#)
- vector.c
  - VECTOR\_MIN\_SIZ, [35](#)
  - vector\_add, [35](#)
  - vector\_at, [36](#)
  - vector\_create, [36](#)
  - vector\_destroy, [36](#)
  - vector\_get\_min\_buf\_siz, [36](#)
  - vector\_min\_siz, [37](#)
  - vector\_resize\_buffer, [37](#)
  - vector\_set\_elem\_at, [37](#)
  - vector\_set\_min\_buf\_siz, [37](#)
- vector.h
  - vector\_add, [25](#)
  - vector\_at, [25](#)
  - vector\_create, [25](#)
  - vector\_destroy, [25](#)
  - vector\_get\_min\_buf\_siz, [26](#)
  - vector\_resize\_buffer, [26](#)
  - vector\_set\_elem\_at, [26](#)
  - vector\_set\_min\_buf\_siz, [26](#)
  - vector\_t, [25](#)
- vector\_add
  - vector.c, [35](#)
  - vector.h, [25](#)
- vector\_at
  - vector.c, [36](#)
  - vector.h, [25](#)
- vector\_create
  - vector.c, [36](#)
  - vector.h, [25](#)
- vector\_destroy
  - vector.c, [36](#)
  - vector.h, [25](#)
- vector\_get\_min\_buf\_siz
  - vector.c, [36](#)
  - vector.h, [26](#)
- vector\_min\_siz
  - vector.c, [37](#)
- vector\_resize\_buffer
  - vector.c, [37](#)
  - vector.h, [26](#)
- vector\_set\_elem\_at
  - vector.c, [37](#)
  - vector.h, [26](#)
- vector\_set\_min\_buf\_siz
  - vector.c, [37](#)
  - vector.h, [26](#)
- vector\_t, [11](#)
  - buffer\_size, [11](#)
  - data, [11](#)
  - member\_size, [11](#)
  - size, [11](#)
  - vector.h, [25](#)