

## Queue

Generated by Doxygen 1.8.11

## Contents

<b>1</b>	<b>Todo List</b>	<b>1</b>
<b>2</b>	<b>Data Structure Index</b>	<b>1</b>
2.1	Data Structures . . . . .	1
<b>3</b>	<b>File Index</b>	<b>2</b>
3.1	File List . . . . .	2
<b>4</b>	<b>Data Structure Documentation</b>	<b>2</b>
4.1	queue_t Struct Reference . . . . .	2
4.1.1	Detailed Description . . . . .	2
4.1.2	Field Documentation . . . . .	2
<b>5</b>	<b>File Documentation</b>	<b>3</b>
5.1	queue.c File Reference . . . . .	3
5.1.1	Detailed Description . . . . .	4
5.1.2	Function Documentation . . . . .	4
5.2	queue.c . . . . .	7
5.3	queue.h File Reference . . . . .	8
5.3.1	Detailed Description . . . . .	9
5.3.2	Function Documentation . . . . .	9
5.4	queue.h . . . . .	12
	<b>Index</b>	<b>15</b>

## 1 Todo List

### Class `queue_t`

`base` should'nt be accessible, see <https://stackoverflow.com/questions/5368028/how-to-make-struct->

## 2 Data Structure Index

### 2.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">queue_t</a>	<a href="#">2</a>
-------------------------	-------------------

## 3 File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

<a href="#">queue.c</a>	
Queue's basic operations implementation (using array)	<a href="#">3</a>
<a href="#">queue.h</a>	
Queue (using array) definition and basic operations	<a href="#">8</a>

## 4 Data Structure Documentation

### 4.1 queue\_t Struct Reference

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

#### Data Fields

- `size_t width`
- `int front`
- `int count`
- `void ** base`
- `int max_size`

#### 4.1.1 Detailed Description

Abstract queue using array.

**Todo** `base` should'nt be accessible, see <https://stackoverflow.com/questions/5368028/how-to-make-struct-member-accessible>

Definition at line [22](#) of file [queue.h](#).

#### 4.1.2 Field Documentation

##### 4.1.2.1 void\*\* base

pointer to the array

Definition at line [26](#) of file [queue.h](#).

#### 4.1.2.2 int count

count element amount

Definition at line 25 of file [queue.h](#).

#### 4.1.2.3 int front

front element index

Definition at line 24 of file [queue.h](#).

#### 4.1.2.4 int max\_size

width \* max\_size bytes is reserved for the queue

Definition at line 27 of file [queue.h](#).

#### 4.1.2.5 size\_t width

element size (in bytes)

Definition at line 23 of file [queue.h](#).

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

- [queue.h](#)

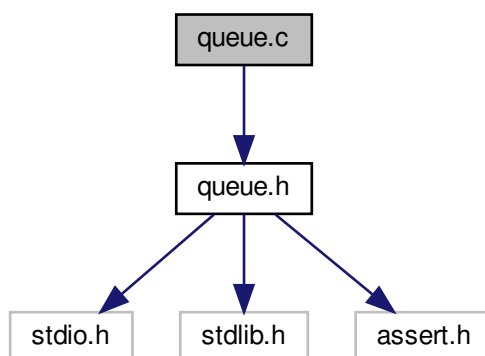
## 5 File Documentation

### 5.1 queue.c File Reference

queue's basic operations implementation (using array)

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

Include dependency graph for queue.c:



## Functions

- int [queue\\_isempty](#) ([queue\\_t](#) \*q)
- int [queue\\_isfull](#) ([queue\\_t](#) \*q)
- void [queue\\_enqueue](#) ([queue\\_t](#) \*q, void \*e)
- void \* [queue\\_dequeue](#) ([queue\\_t](#) \*q)
- [queue\\_t](#) \* [queue\\_create](#) (size\_t width, int max\_size)
- void [queue\\_destruct](#) ([queue\\_t](#) \*q)
- void [queue\\_int\\_print](#) ([queue\\_t](#) \*q)

### 5.1.1 Detailed Description

queue's basic operations implementation (using array)

#### Author

Firmin MARTIN

#### Version

0.1

#### Date

28/12/2017

Definition in file [queue.c](#).

### 5.1.2 Function Documentation

#### 5.1.2.1 [queue\\_t](#)\* [queue\\_create](#) ( [size\\_t](#) width, int max\_size )

Given the size of each element and the queue size, create a queue.

#### Parameters

<i>width</i>	size of each element
<i>max_size</i>	size of the queue, max_size*width bytes will be reserved (definitively) for the queue

#### Returns

a queue initialized

Definition at line 70 of file [queue.c](#).

```
00070                                     {
00071     queue\_t* q = malloc(sizeof(queue\_t));
00072     assert(q);
00073     q->width = width;
```

```
00074     q->max_size = max_size ;
00075     q->base = (void**) calloc(q->max_size, sizeof(void*));
00076     assert(q->base);
00077     q->front = 0;
00078     q->count = 0;
00079     return q;
00080 }
```

#### 5.1.2.2 void\* queue\_dequeue( queue\_t \* q )

Dequeue an element from the queue s.

##### Parameters

<i>q</i>	queue
----------	-------

##### Returns

an element or NULL if the queue is empty

Definition at line 53 of file [queue.c](#).

```
00053     {
00054     if(queue_isempty(q)) {
00055         fprintf(stderr, "The queue is empty : failed to dequeue.\n");
00056         return NULL;
00057     }
00058     void* e = q->base[(q->front - q->count + q->max_size)%q->
00059 max_size];
00059     q->count--;
00060     return e;
00061 }
```

Here is the call graph for this function:



#### 5.1.2.3 void queue\_destruct( queue\_t \* q )

Free a queue.

##### Parameters

<i>q</i>	a queue
----------	---------

Definition at line 87 of file [queue.c](#).

```

00087             {
00088         for(int i=0; i<q->max_size;i++){
00089             free(q->base[i]);
00090         }
00091         free(q->base);
00092         free(q);
00093     }

```

#### 5.1.2.4 void queue\_enqueue ( queue\_t \* q, void \* e )

Enqueue an element e into the queue q.

##### Parameters

<i>q</i>	queue
<i>e</i>	element which be enqueued

Definition at line 37 of file [queue.c](#).

```

00037             {
00038         if(queue_isfull(q)) {
00039             fprintf(stderr, "The queue is full : failed to enqueue.\n");
00040             return;
00041         }
00042         q->base[q->front] = e;
00043         q->count++;
00044         q->front = (q->front+ 1)%q->max_size;
00045     }

```

Here is the call graph for this function:



#### 5.1.2.5 void queue\_int\_print ( queue\_t \* q )

Print an int queue

Definition at line 100 of file [queue.c](#).

```

00100             {
00101         for(int i = 0; i < q->max_size; i++) {
00102             if ((q->front - i + q->max_size) % q->max_size <= q->
count && i != q->front)
00103                 printf("[%d]", *((int*)q->base[i]));
00104             else
00105                 printf("[ ]");
00106         }
00107         puts("");
00108     }

```

#### 5.1.2.6 int queue\_isempty ( queue\_t \* q )

Determinate the emptiness of a queue.

## Parameters

s	queue
---	-------

## Returns

1 if the queue s is empty, 0 otherwise.

Definition at line 17 of file [queue.c](#).

```
00017 {
00018     return q->count == 0;
00019 }
```

## 5.1.2.7 int queue\_isfull ( queue\_t \* q )

Determinate the fullness of a queue.

## Parameters

s	queue
---	-------

## Returns

1 if the queue s is full, 0 otherwise.

Definition at line 27 of file [queue.c](#).

```
00027 {
00028     return q->count == q->max_size;
00029 }
```

## 5.2 queue.c

```
00001
00009 #include "queue.h"
00010
00017 int queue_isempty(queue_t* q) {
00018     return q->count == 0;
00019 }
00020
00027 int queue_isfull(queue_t* q) {
00028     return q->count == q->max_size;
00029 }
00030
00037 void queue_enqueue(queue_t* q, void* e) {
00038     if(queue_isfull(q)) {
00039         fprintf(stderr, "The queue is full : failed to enqueue.\n");
00040         return;
00041     }
00042     q->base[q->front] = e;
00043     q->count++;
00044     q->front = (q->front + 1) % q->max_size;
00045 }
00046
00053 void* queue_dequeue(queue_t* q) {
00054     if(queue_isempty(q)) {
00055         fprintf(stderr, "The queue is empty : failed to dequeue.\n");
00056         return NULL;
00057     }
00058 }
```



```

00058     void* e = q->base[(q->front - q->count + q->max_size)%q->
max_size];
00059     q->count--;
00060     return e;
00061 }
00062
00070 queue_t* queue_create(size_t width, int max_size) {
00071     queue_t* q = malloc(sizeof(queue_t));
00072     assert(q);
00073     q->width = width;
00074     q->max_size = max_size ;
00075     q->base = (void**) calloc(q->max_size, sizeof(void*));
00076     assert(q->base);
00077     q->front = 0;
00078     q->count = 0;
00079     return q;
00080 }
00081
00087 void queue_destruct(queue_t* q) {
00088     for(int i=0; i<q->max_size;i++){
00089         free(q->base[i]);
00090     }
00091     free(q->base);
00092     free(q);
00093 }
00094
00095
00100 void queue_int_print(queue_t* q) {
00101     for(int i = 0; i < q->max_size; i++) {
00102         if ((q->front - i + q->max_size) % q->max_size <= q->
count && i != q->front)
00103             printf("[%d]", *((int*)q->base[i]));
00104         else
00105             printf("[ ]");
00106     }
00107     puts("");
00108 }
00109

```

### 5.3 queue.h File Reference

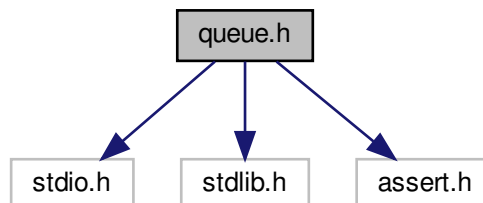
queue (using array) definition and basic operations

```

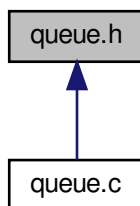
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

```

Include dependency graph for queue.h:



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



#### Data Structures

- struct [queue\\_t](#)

#### Functions

- [queue\\_t \\* queue\\_create](#) (size\_t width, int max\_size)
- void [queue\\_destruct](#) (queue\_t \*q)
- int [queue\\_isempty](#) (queue\_t \*q)
- int [queue\\_isfull](#) (queue\_t \*q)
- void \* [queue\\_dequeue](#) (queue\_t \*q)
- void [queue\\_enqueue](#) (queue\_t \*q, void \*e)
- void [queue\\_int\\_print](#) (queue\_t \*q)

#### 5.3.1 Detailed Description

queue (using array) definition and basic operations

##### Author

Firmin MARTIN

##### Version

0.1

##### Date

28/12/2017

Definition in file [queue.h](#).

#### 5.3.2 Function Documentation

##### 5.3.2.1 [queue\\_t\\*](#) queue\_create ( size\_t width, int max\_size )

Given the size of each element and the queue size, create a queue.

**Parameters**

<i>width</i>	size of each element
<i>max_size</i>	size of the queue, <code>max_size*width</code> bytes will be reserved (definitively) for the queue

**Returns**

a queue initialized

Definition at line 70 of file [queue.c](#).

```

00070                                     {
00071     queue_t* q = malloc(sizeof(queue_t));
00072     assert(q);
00073     q->width = width;
00074     q->max_size = max_size ;
00075     q->base = (void**) calloc(q->max_size, sizeof(void*));
00076     assert(q->base);
00077     q->front = 0;
00078     q->count = 0;
00079     return q;
00080 }
```

**5.3.2.2 void\* queue\_dequeue ( queue\_t \* q )**

Dequeue an element from the queue s.

**Parameters**

<i>q</i>	queue
----------	-------

**Returns**

an element or NULL if the queue is empty

Definition at line 53 of file [queue.c](#).

```

00053                                     {
00054     if(queue_isempty(q)) {
00055         fprintf(stderr, "The queue is empty : failed to dequeue.\n");
00056         return NULL;
00057     }
00058     void* e = q->base[(q->front - q->count + q->max_size)%q->
max_size];
00059     q->count--;
00060     return e;
00061 }
```

Here is the call graph for this function:



## 5.3.2.3 void queue\_destruct ( queue\_t \* q )

Free a queue.

## Parameters

<i>q</i>	a queue
----------	---------

Definition at line 87 of file [queue.c](#).

```

00087                                     {
00088     for(int i=0; i<q->max_size;i++){
00089         free(q->base[i]);
00090     }
00091     free(q->base);
00092     free(q);
00093 }
```

## 5.3.2.4 void queue\_enqueue ( queue\_t \* q, void \* e )

Enqueue an element e into the queue q.

## Parameters

<i>q</i>	queue
<i>e</i>	element which be enqueued

Definition at line 37 of file [queue.c](#).

```

00037                                     {
00038     if(queue_isfull(q)) {
00039         fprintf(stderr, "The queue is full : failed to enqueue.\n");
00040         return;
00041     }
00042     q->base[q->front] = e;
00043     q->count++;
00044     q->front = (q->front+ 1)%q->max_size;
00045 }
```

Here is the call graph for this function:



## 5.3.2.5 void queue\_int\_print ( queue\_t \* q )

Print an int queue

Definition at line 100 of file [queue.c](#).

```

00100      {
00101      for(int i = 0; i < q->max_size; i++) {
00102          if ((q->front - i + q->max_size) % q->max_size <= q->
count && i != q->front)
00103              printf("[%d]", *((int*)q->base[i]));
00104          else
00105              printf("[]");
00106      }
00107      puts("");
00108  }

```

### 5.3.2.6 int queue\_isempty ( queue\_t \* q )

Determinate the emptiness of a queue.

#### Parameters

s	queue
---	-------

#### Returns

1 if the queue s is empty, 0 otherwise.

Definition at line 17 of file [queue.c](#).

```

00017      {
00018      return q->count == 0;
00019  }

```

### 5.3.2.7 int queue\_isfull ( queue\_t \* q )

Determinate the fullness of a queue.

#### Parameters

s	queue
---	-------

#### Returns

1 if the queue s is full, 0 otherwise.

Definition at line 27 of file [queue.c](#).

```

00027      {
00028      return q->count == q->max_size;
00029  }

```

## 5.4 queue.h

```

00001 #ifndef QUEUE_H
00002 #define QUEUE_H
00003
00004 #include <stdio.h>
00005 #include <stdlib.h>

```

```
00006 #include <assert.h>
00007
00022 typedef struct {
00023     size_t width;
00024     int front;
00025     int count;
00026     void** base;
00027     int max_size;
00028 } queue_t;
00029
00030 queue_t* queue_create(size_t width, int max_size);
00031 void queue_destruct(queue_t* q);
00032 int queue_isempty(queue_t* q);
00033 int queue_isfull(queue_t* q);
00034 void* queue_dequeue(queue_t* q);
00035 void queue_enqueue(queue_t* q, void* e);
00036 void queue_int_print(queue_t* q);
00037
00038 #endif /* ifndef QUEUE_H */
```



## Index

base [queue\\_t, 2](#)

count [queue\\_t, 2](#)

front [queue\\_t, 3](#)

max\_size [queue\\_t, 3](#)

queue.c, [3](#)

- [queue\\_create, 4](#)
- [queue\\_dequeue, 5](#)
- [queue\\_destruct, 5](#)
- [queue\\_enqueue, 6](#)
- [queue\\_int\\_print, 6](#)
- [queue\\_isempty, 6](#)
- [queue\\_isfull, 7](#)

queue.h, [8](#)

- [queue\\_create, 9](#)
- [queue\\_dequeue, 10](#)
- [queue\\_destruct, 10](#)
- [queue\\_enqueue, 11](#)
- [queue\\_int\\_print, 11](#)
- [queue\\_isempty, 12](#)
- [queue\\_isfull, 12](#)

queue\_create

- [queue.c, 4](#)
- [queue.h, 9](#)

queue\_dequeue

- [queue.c, 5](#)
- [queue.h, 10](#)

queue\_destruct

- [queue.c, 5](#)
- [queue.h, 10](#)

queue\_enqueue

- [queue.c, 6](#)
- [queue.h, 11](#)

queue\_int\_print

- [queue.c, 6](#)
- [queue.h, 11](#)

queue\_isempty

- [queue.c, 6](#)
- [queue.h, 12](#)

queue\_isfull

- [queue.c, 7](#)
- [queue.h, 12](#)

queue\_t, [2](#)

- [base, 2](#)
- [count, 2](#)
- [front, 3](#)
- [max\\_size, 3](#)
- [width, 3](#)

width

[queue\\_t, 3](#)