

Deque

Generated by Doxygen 1.8.11

Contents

1	Todo List	1
2	Data Structure Index	1
2.1	Data Structures	1
3	File Index	2
3.1	File List	2
4	Data Structure Documentation	2
4.1	deque_t Struct Reference	2
4.1.1	Detailed Description	2
4.1.2	Field Documentation	2
5	File Documentation	3
5.1	deque.c File Reference	3
5.1.1	Detailed Description	4
5.1.2	Function Documentation	4
5.2	deque.c	9
5.3	deque.h File Reference	10
5.3.1	Detailed Description	11
5.3.2	Function Documentation	11
5.4	deque.h	15
	Index	17

1 Todo List

Class deque_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:

deque_t	2
-------------------------	---

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

deque.c	
Deque's basic operations implementation (using array)	3
deque.h	
Deque (using array) definition and basic operations	10

4 Data Structure Documentation

4.1 deque_t Struct Reference

```
#include <deque.h>
```

Data Fields

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

4.1.1 Detailed Description

Abstract deque 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 [deque.h](#).

4.1.2 Field Documentation

4.1.2.1 `void** base`

pointer to the array

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

4.1.2.2 int count

count element amount

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

4.1.2.3 int front

front element index

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

4.1.2.4 int max_size

width * max_size bytes is reserved for the deque

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

4.1.2.5 size_t width

element size (in bytes)

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

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

- [deque.h](#)

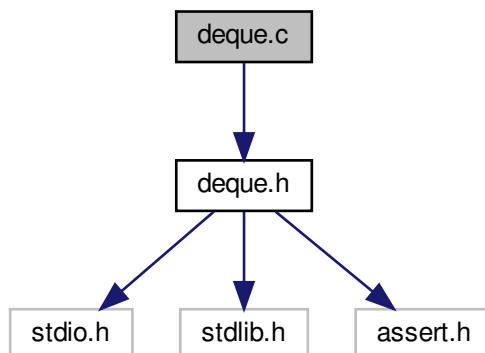
5 File Documentation

5.1 deque.c File Reference

deque's basic operations implementation (using array)

```
#include "deque.h"
```

Include dependency graph for deque.c:



Functions

- int [deque_isempty](#) ([deque_t](#) *q)
- int [deque_isfull](#) ([deque_t](#) *q)
- void [deque_enqueue](#) ([deque_t](#) *q, void *e)
- void [deque_enqueue_other_end](#) ([deque_t](#) *q, void *e)
- void * [deque_dequeue](#) ([deque_t](#) *q)
- void * [deque_dequeue_other_end](#) ([deque_t](#) *q)
- [deque_t](#) * [deque_create](#) (size_t width, int max_size)
- void [deque_destruct](#) ([deque_t](#) *q)
- void [deque_int_print](#) ([deque_t](#) *q)

5.1.1 Detailed Description

deque's basic operations implementation (using array)

Author

Firmin MARTIN

Version

0.1

Date

28/12/2017

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

5.1.2 Function Documentation

5.1.2.1 [deque_t](#)* [deque_create](#) ([size_t](#) width, int max_size)

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

Parameters

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

Returns

a deque initialized

Definition at line 101 of file [deque.c](#).

00101

{

```

00102     deque_t* q = malloc(sizeof(deque_t));
00103     assert(q);
00104     q->width = width;
00105     q->max_size = max_size ;
00106     q->base = (void**) calloc(q->max_size, sizeof(void*));
00107     assert(q->base);
00108     q->front = 0;
00109     q->count = 0;
00110     return q;
00111 }

```

5.1.2.2 void* deque_dequeue (deque_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 67 of file [deque.c](#).

```

00067     {
00068     if(deque_isempty(q)) {
00069         fprintf(stderr, "The deque is empty : failed to dequeue.\n");
00070         return NULL;
00071     }
00072     void* e = q->base[(q->front - q->count + q->max_size) % q->
max_size];
00073     q->count--;
00074     return e;
00075 }

```

Here is the call graph for this function:



5.1.2.3 void* deque_dequeue_other_end (deque_t * q)

Dequeue an element from the deque s at the other end.

Parameters

<i>q</i>	deque
----------	-------

Returns

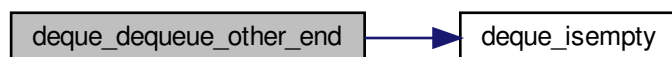
an element or NULL if the deque is empty

Definition at line 83 of file [deque.c](#).

```

00083                                     {
00084     if(deque_isempty(q)) {
00085         fprintf(stderr, "The deque is empty : failed to dequeue.\n");
00086         return NULL;
00087     }
00088     q->front = (q->front - 1 + q->max_size) % q->max_size;
00089     void* e = q->base[q->front];
00090     q->count--;
00091     return e;
00092 }
```

Here is the call graph for this function:



5.1.2.4 void deque_destruct (deque_t * q)

Free a deque.

Parameters

<i>q</i>	a deque
----------	---------

Definition at line 118 of file [deque.c](#).

```

00118                                     {
00119     free(q->base);
00120     free(q);
00121 }
```

5.1.2.5 void deque_enqueue (deque_t * q, void * e)

Enqueue an element e into the deque q.

Parameters

<i>q</i>	deque
<i>e</i>	element which be endequeued

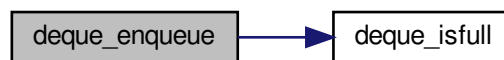
Definition at line 37 of file [deque.c](#).

```

00037                                     {
00038     if(deque_isfull(q)) {
00039         fprintf(stderr, "The deque 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:



5.1.2.6 void deque_enqueue_other_end (deque_t * q, void * e)

Enqueue an element *e* into the deque *q* at the other end.

Parameters

<i>q</i>	deque
<i>e</i>	element which be ended

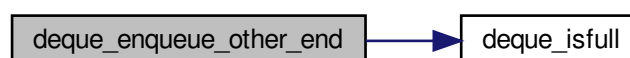
Definition at line 53 of file [deque.c](#).

```

00053                                     {
00054     if(deque_isfull(q)) {
00055         fprintf(stderr, "The deque is full : failed to enqueue.\n");
00056         return;
00057     } q->base[(q->front - (q->count + 1) + q->max_size) % q->
max_size] = e;
00058     q->count++;
00059 }

```

Here is the call graph for this function:



5.1.2.7 void deque_int_print (deque_t * q)

Print an int deque

Definition at line 127 of file [deque.c](#).

```

00127     {
00128         for(int i = 0; i < q->max_size; i++) {
00129             if ((q->front - i + q->max_size) % q->max_size <= q->
count && i != q->front)
00130                 printf("[%d]", *((int*)q->base[i]));
00131             else
00132                 printf("[ ]");
00133         }
00134         puts("");
00135     }

```

5.1.2.8 int deque_isempty (deque_t * q)

Determinate the emptiness of a deque.

Parameters

s	deque
---	-------

Returns

1 if the deque s is empty, 0 otherwise.

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

```

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

```

5.1.2.9 int deque_isfull (deque_t * q)

Determinate the fullness of a deque.

Parameters

s	deque
---	-------

Returns

1 if the deque s is full, 0 otherwise.

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

```

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

```

5.2 deque.c

```

00001
00009 #include "deque.h"
00010
00017 int deque_isempty(deque_t* q) {
00018     return q->count == 0;
00019 }
00020
00027 int deque_isfull(deque_t* q) {
00028     return q->count == q->max_size;
00029 }
00030
00037 void deque_enqueue(deque_t* q, void* e) {
00038     if(deque_isfull(q)) {
00039         fprintf(stderr, "The deque 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 deque_enqueue_other_end(deque_t* q, void* e) {
00054     if(deque_isfull(q)) {
00055         fprintf(stderr, "The deque is full : failed to enqueue.\n");
00056         return;
00057     } q->base[(q->front - (q->count + 1) + q->max_size) % q->
max_size] = e;
00058     q->count++;
00059 }
00060
00067 void* deque_dequeue(deque_t* q) {
00068     if(deque_isempty(q)) {
00069         fprintf(stderr, "The deque is empty : failed to dequeue.\n");
00070         return NULL;
00071     }
00072     void* e = q->base[(q->front - q->count + q->max_size) % q->
max_size];
00073     q->count--;
00074     return e;
00075 }
00076
00083 void* deque_dequeue_other_end(deque_t* q) {
00084     if(deque_isempty(q)) {
00085         fprintf(stderr, "The deque is empty : failed to dequeue.\n");
00086         return NULL;
00087     }
00088     q->front = (q->front - 1 + q->max_size) % q->max_size;
00089     void* e = q->base[q->front];
00090     q->count--;
00091     return e;
00092 }
00093
00101 deque_t* deque_create(size_t width, int max_size) {
00102     deque_t* q = malloc(sizeof(deque_t));
00103     assert(q);
00104     q->width = width;
00105     q->max_size = max_size;
00106     q->base = (void**) calloc(q->max_size, sizeof(void*));
00107     assert(q->base);
00108     q->front = 0;
00109     q->count = 0;
00110     return q;
00111 }
00112
00118 void deque_destruct(deque_t* q) {
00119     free(q->base);
00120     free(q);
00121 }
00122
00127 void deque_int_print(deque_t* q) {
00128     for(int i = 0; i < q->max_size; i++) {
00129         if ((q->front - i + q->max_size) % q->max_size <= q->
count && i != q->front)
00130             printf("[%d]", *((int*)q->base[i]));
00131         else
00132             printf("[ ]");
00133     }
00134     puts("");
00135 }
00136

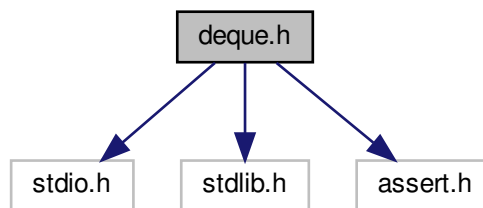
```

5.3 deque.h File Reference

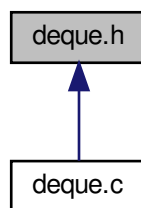
deque (using array) definition and basic operations

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

Include dependency graph for deque.h:



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



Data Structures

- struct [deque_t](#)

Functions

- [deque_t * deque_create](#) (size_t width, int max_size)
- void [deque_destruct](#) (deque_t *q)
- int [deque_isempty](#) (deque_t *q)
- int [deque_isfull](#) (deque_t *q)
- void * [deque_dequeue](#) (deque_t *q)
- void * [deque_dequeue_other_end](#) (deque_t *q)
- void [deque_enqueue_other_end](#) (deque_t *q, void *e)
- void [deque_enqueue](#) (deque_t *q, void *e)
- void [deque_int_print](#) (deque_t *q)

5.3.1 Detailed Description

deque (using array) definition and basic operations

Author

Firmin MARTIN

Version

0.1

Date

28/12/2017

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

5.3.2 Function Documentation

5.3.2.1 deque_t* deque_create (size_t width, int max_size)

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

Parameters

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

Returns

a deque initialized

Definition at line 101 of file [deque.c](#).

```
00101     {
00102         deque_t* q = malloc(sizeof(deque_t));
00103         assert(q);
00104         q->width = width;
00105         q->max_size = max_size ;
00106         q->base = (void**) calloc(q->max_size, sizeof(void*));
00107         assert(q->base);
00108         q->front = 0;
00109         q->count = 0;
00110         return q;
00111     }
```

5.3.2.2 void* deque_dequeue (deque_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 67 of file [deque.c](#).

```

00067             {
00068         if(deque_isempty(q)) {
00069             fprintf(stderr, "The deque is empty : failed to dequeue.\n");
00070             return NULL;
00071         }
00072         void* e = q->base[(q->front - q->count + q->max_size) % q->
max_size];
00073         q->count--;
00074         return e;
00075     }

```

Here is the call graph for this function:



5.3.2.3 void* deque_dequeue_other_end(deque_t* q)

Dequeue an element from the deque *s* at the other end.

Parameters

<i>q</i>	deque
----------	-------

Returns

an element or NULL if the deque is empty

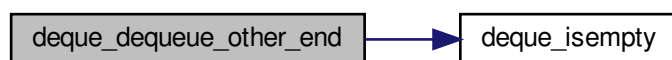
Definition at line 83 of file [deque.c](#).

```

00083             {
00084         if(deque_isempty(q)) {
00085             fprintf(stderr, "The deque is empty : failed to dequeue.\n");
00086             return NULL;
00087         }
00088         q->front = (q->front - 1 + q->max_size) % q->max_size;
00089         void* e = q->base[q->front];
00090         q->count--;
00091         return e;
00092     }

```

Here is the call graph for this function:



5.3.2.4 void deque_destruct (deque_t * q)

Free a deque.

Parameters

<i>q</i>	a deque
----------	---------

Definition at line 118 of file [deque.c](#).

```

00118                                     {
00119     free(q->base);
00120     free(q);
00121 }
```

5.3.2.5 void deque_enqueue (deque_t * q, void * e)

Enqueue an element e into the deque q.

Parameters

<i>q</i>	deque
<i>e</i>	element which be endequeued

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

```

00037                                     {
00038     if(deque_isfull(q)) {
00039         fprintf(stderr, "The deque 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:



5.3.2.6 void deque_enqueue_other_end (deque_t * q, void * e)

Enqueue an element e into the deque q at the other end.

Parameters

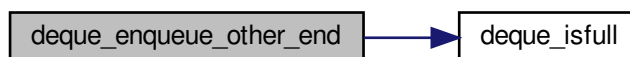
<i>q</i>	deque
<i>e</i>	element which be ended

Definition at line 53 of file deque.c.

```

00053                                     {
00054     if (deque_isfull(q)) {
00055         fprintf(stderr, "The deque is full : failed to enqueue.\n");
00056         return;
00057     } q->base[(q->front - (q->count + 1) + q->max_size) % q->
max_size] = e;
00058     q->count++;
00059 }
  
```

Here is the call graph for this function:



5.3.2.7 void deque_int_print (deque_t * q)

Print an int deque

Definition at line 127 of file deque.c.

```

00127                                     {
00128     for(int i = 0; i < q->max_size; i++) {
00129         if ((q->front - i + q->max_size) % q->max_size <= q->
count && i != q->front)
00130             printf("[%d]", *((int*)q->base[i]));
00131         else
00132             printf("[ ]");
00133     }
00134     puts("");
00135 }
  
```

5.3.2.8 int deque_isempty (deque_t * q)

Determinate the emptiness of a deque.

Parameters

s	deque
---	-------

Returns

1 if the deque s is empty, 0 otherwise.

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

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

5.3.2.9 int deque_isfull (deque_t * q)

Determinate the fullness of a deque.

Parameters

s	deque
---	-------

Returns

1 if the deque s is full, 0 otherwise.

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

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

5.4 deque.h

```
00001 #ifndef DEQUE_H
00002 #define DEQUE_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 } deque_t;
00029
00030 deque_t* deque_create(size_t width, int max_size);
00031 void deque_destruct(deque_t* q);
00032 int deque_isempty(deque_t* q);
00033 int deque_isfull(deque_t* q);
```



```
00034 void* deque_dequeue(deque_t* q);
00035 void* deque_dequeue_other_end(deque_t* q);
00036 void deque_enqueue_other_end(deque_t* q, void* e);
00037 void deque_enqueue(deque_t* q, void* e);
00038 void deque_int_print(deque_t* q);
00039
00040 #endif /* ifndef deque_H*/
```

Index

- base
 - [deque_t, 2](#)
- count
 - [deque_t, 2](#)
- deque.c, [3](#)
 - [deque_create, 4](#)
 - [deque_dequeue, 5](#)
 - [deque_dequeue_other_end, 5](#)
 - [deque_destruct, 6](#)
 - [deque_enqueue, 6](#)
 - [deque_enqueue_other_end, 7](#)
 - [deque_int_print, 7](#)
 - [deque_isempty, 8](#)
 - [deque_isfull, 8](#)
- deque.h, [10](#)
 - [deque_create, 11](#)
 - [deque_dequeue, 11](#)
 - [deque_dequeue_other_end, 12](#)
 - [deque_destruct, 13](#)
 - [deque_enqueue, 13](#)
 - [deque_enqueue_other_end, 14](#)
 - [deque_int_print, 14](#)
 - [deque_isempty, 14](#)
 - [deque_isfull, 15](#)
- deque_create
 - [deque.c, 4](#)
 - [deque.h, 11](#)
- deque_dequeue
 - [deque.c, 5](#)
 - [deque.h, 11](#)
- deque_dequeue_other_end
 - [deque.c, 5](#)
 - [deque.h, 12](#)
- deque_destruct
 - [deque.c, 6](#)
 - [deque.h, 13](#)
- deque_enqueue
 - [deque.c, 6](#)
 - [deque.h, 13](#)
- deque_enqueue_other_end
 - [deque.c, 7](#)
 - [deque.h, 14](#)
- deque_int_print
 - [deque.c, 7](#)
 - [deque.h, 14](#)
- deque_isempty
 - [deque.c, 8](#)
 - [deque.h, 14](#)
- deque_isfull
 - [deque.c, 8](#)
 - [deque.h, 15](#)
- deque_t, [2](#)
 - [base, 2](#)
 - [count, 2](#)
 - [front, 3](#)
 - [max_size, 3](#)
 - [width, 3](#)
- front
 - [deque_t, 3](#)
- max_size
 - [deque_t, 3](#)
- width
 - [deque_t, 3](#)