Queue

Generated by Doxygen 1.8.11

Contents

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:

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

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-struc`

Definition at line 21 of file queue.h.

4.1.2 Field Documentation

4.1.2.1 void** base

pointer to the array

Definition at line 25 of file queue.h.

4.1.2.2 int count

count element amount

Definition at line 24 of file queue.h.

4.1.2.3 int front

front element index

Definition at line 23 of file queue.h.

4.1.2.4 int max_size

width ∗ max_size bytes is reserved for the queue

Definition at line 26 of file queue.h.

4.1.2.5 size_t width

element size (in bytes)

Definition at line 22 of file queue.h.

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

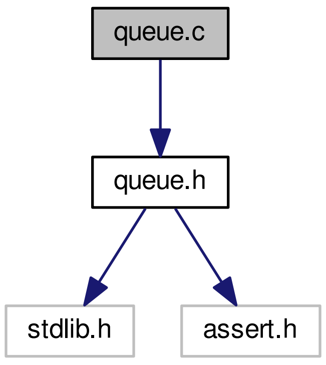
- queue.h

5 File Documentation

5.1 queue.c File Reference

queue's basic operations implementation (using dynamic array)

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



Functions

- int queue_isempty (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)

5.1.1 Detailed Description

queue's basic operations implementation (using dynamic 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

width	size of each element
max_size	size of the queue, max_size∗width bytes will be reserved (definitively) for the queue

Returns

a queue initialized

Note

This queue implementation assume that the amount of element will never exceed max_size. See queue_↩ enqueue for more information on the behavior in the excess case.

Definition at line 60 of file queue.c.

```
00060                                                         {
00061     queue_t* q = malloc(sizeof(queue_t));
00062     assert(q);
00063     q->width = width;
00064     q->max_size = max_size ;
00065     q->base = (void**) calloc(q->max_size, sizeof(void*));
00066     assert(q->base);
00067     q->front = 0;
00068     q->count = 0;
00069     return q;
00070 }
```

5.1.2.2 void∗ queue_dequeue (queue_t ∗ *q*)

Dequeue an element from the queue s.

Parameters

q	queue

Returns

an element

Definition at line 43 of file queue.c.

```
00043                                   {
00044     void* e = q->base[(q->front - q->count + q->max_size)%q->
      max_size];
00045     q->count--;
00046     return e;
00047 }
```

5.1.2.3 void queue_destruct (queue_t ∗ *q*)

Free a queue.

Parameters

q	a queue

Definition at line 77 of file queue.c.

```
00077                                   {
00078     free(q->base);
00079     free(q);
00080 }
```

5.1.2.4 void queue_enqueue (queue_t ∗ *q*, void ∗ *e*)

Enqueue an element e into the queue q.

Parameters

q	queue
e	element which be enqueued

Note

Note that if the max size is reached, this function will overwrite the queue and consider the queue as empty, i.e. at the end the queue has just one element.

Definition at line 30 of file queue.c.

```
00030                                          {
00031     q->base[q->front] = e;
00032     if (q->front == q->max_size - 1) q->front = 0;
00033     else q->front++;
00034     q->count = (q->count + 1) % q->max_size;
00035 }
```

5.1.2.5 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 }
```

5.2 queue.c

```
00001
00009 #include "queue.h"
00010
00017 int queue_isempty(queue_t* q) {
00018     return q->count == 0;
00019 }
00020
00030 void queue_enqueue(queue_t* q, void* e) {
00031     q->base[q->front] = e;
00032     if (q->front == q->max_size - 1) q->front = 0;
00033     else q->front++;
00034     q->count = (q->count + 1) % q->max_size;
00035 }
00036
00043 void* queue_dequeue(queue_t* q) {
00044     void* e = q->base[(q->front - q->count + q->max_size)%q->
      max_size];
00045     q->count--;
00046     return e;
00047 }
00048
00060 queue_t* queue_create(size_t width, int max_size) {
00061     queue_t* q = malloc(sizeof(queue_t));
00062     assert(q);
00063     q->width = width;
00064     q->max_size = max_size ;
00065     q->base = (void**) calloc(q->max_size, sizeof(void*));
00066     assert(q->base);
00067     q->front = 0;
00068     q->count = 0;
```

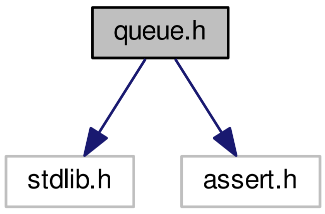
```
    return q;
}

void queue_destruct(queue_t* q) {
    free(q->base);
    free(q);
}



```

5.3 queue.h File Reference

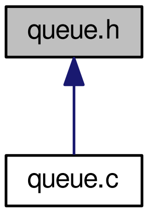
queue (using array) definition and basic operations

```
#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)
- void ∗ queue_dequeue (queue_t ∗q)
- void queue_enqueue (queue_t ∗q, void ∗e)

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

width	size of each element
max_size	size of the queue, max_size∗width bytes will be reserved (definitively) for the queue

Returns

a queue initialized

Note

This queue implementation assume that the amount of element will never exceed max_size. See queue_↩ enqueue for more information on the behavior in the excess case.

Definition at line 60 of file queue.c.

```
00060                                          {
00061     queue_t* q = malloc(sizeof(queue_t));
00062     assert(q);
00063     q->width = width;
00064     q->max_size = max_size ;
00065     q->base = (void**) calloc(q->max_size, sizeof(void*));
00066     assert(q->base);
00067     q->front = 0;
00068     q->count = 0;
00069     return q;
00070 }
```

5.3.2.2 void∗ queue_dequeue (queue_t ∗ *q*)

Dequeue an element from the queue s.

Parameters

q	queue

Returns

an element

Definition at line 43 of file queue.c.

```
00043                             {
00044     void* e = q->base[(q->front - q->count + q->max_size)%q->
     max_size];
00045     q->count--;
00046     return e;
00047 }
```

5.3.2.3 void queue_destruct (queue_t ∗ *q*)

Free a queue.

Parameters

q	a queue

Definition at line 77 of file queue.c.

```
00077                             {
00078     free(q->base);
00079     free(q);
00080 }
```

5.3.2.4 void queue_enqueue (queue_t ∗ *q,* void ∗ *e*)

Enqueue an element e into the queue q.

Parameters

q	queue
e	element which be enqueued

Note

Note that if the max size is reached, this function will overwrite the queue and consider the queue as empty, i.e. at the end the queue has just one element.

Definition at line 30 of file queue.c.

```
00030                                           {
00031     q->base[q->front] = e;
00032     if (q->front == q->max_size - 1) q->front = 0;
00033     else q->front++;
00034     q->count = (q->count + 1) % q->max_size;
00035 }
```

5.3.2.5 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 }
```

5.4 queue.h

```
00001 #ifndef STACK_H
00002 #define STACK_H
00003 
00004 #include <stdlib.h>
00005 #include <assert.h>
00006 
00021 typedef struct {
00022     size_t width;
00023     int front;
00024     int count;
00025     void** base;
00026     int max_size;
00027 } queue_t;
00028 
00029 queue_t* queue_create(size_t width, int max_size);
00030 void queue_destruct(queue_t* q);
00031 int queue_isempty(queue_t* q);
00032 void* queue_dequeue(queue_t* q);
00033 void queue_enqueue(queue_t* q, void* e);
00034 
00035 #endif /* ifndef STACK_H */
```

Index