Stack

Generated by Doxygen 1.8.11

Sun Dec 31 2017 17:48:05

Contents

1 Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

stack_t **2**

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

3 Data Structure Documentation

3.1 stack_t Struct Reference

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

Data Fields

- size_t width
- int top
- void ** base
- int mem_size

3.1.1 Detailed Description

Abstract stack using dynamic array.

Definition at line 20 of file stack.h.

3.1.2 Field Documentation

3.1.2.1 void** base

pointer to the dynamic array

Definition at line 23 of file stack.h.

3.1.2.2 int mem_size

width * mem_size bytes is reserved for the dynamic array

Definition at line 24 of file stack.h.

3.1.2.3 int top

top element index

Definition at line 22 of file stack.h.

3.1.2.4 size_t width

element size (in bytes)

Definition at line 21 of file stack.h.

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

- stack.h

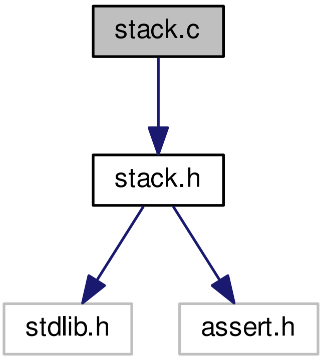
4 File Documentation

4.1 stack.c File Reference

stack's basic operations implementation (using dynamic array)

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

Include dependency graph for stack.c:



Functions

- int stack_isempty (stack_t ∗s)
- void stack_push (stack_t ∗s, void ∗e)
- void ∗ stack_pop (stack_t ∗s)
- stack_t ∗ stack_create (size_t width)
- void stack_destruct (stack_t ∗s)

4.1.1 Detailed Description

stack's basic operations implementation (using dynamic array)

Author

Firmin MARTIN

Version

0.1

Date

28/12/2017

Definition in file stack.c.

4.1.2 Function Documentation

4.1.2.1 stack_t∗ stack_create (size_t *width*)

Given the size of each element, create a stack 10 ∗ sizeof(void∗) bytes is reserved by default.

Parameters

width	size of each element

Returns

a stack initialized

Definition at line 57 of file stack.c.

```
00057                                         {
00058       stack_t* s = malloc(sizeof(stack_t));
00059       assert(s);
00060       s->width = width;
00061       s->mem_size = 10;
00062       s->base = (void**) malloc(sizeof(void*) * s->mem_size);
00063       assert(s->base);
00064       s->top = -1;
00065       return s;
00066 }
```

4.1.2.2 void stack_destruct (stack_t ∗ *s*)

Free a stack.

Parameters

s	a stack

Definition at line 73 of file stack.c.

```
00073                                {
00074     free(s->base);
00075     free(s);
00076 }
```

4.1.2.3 int stack_isempty (stack_t ∗ *s*)

Determinate the emptiness of a stack.

Parameters

s	stack

Returns

1 if the stack s is empty, 0 otherwise.

Definition at line 17 of file stack.c.

```
00017                              {
00018     return s->top == -1;
00019 }
```

4.1.2.4 void∗ stack_pop (stack_t ∗ *s*)

Pop out an element from the stack s.

Parameters

s	stack

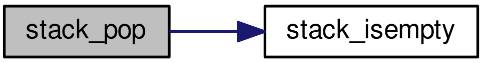
Returns

an element

Definition at line 44 of file stack.c.

```
00044                            {
00045     if (stack_isempty(s)) return NULL;
00046     s->top--;
00047     return s->base[s->top + 1];
00048 }
```

Here is the call graph for this function:



4.1.2.5 void stack_push (stack_t ∗ *s,* void ∗ *e*)

Push an element e into the stack s.

Parameters

s	stack
e	element which be pushed

Definition at line 27 of file stack.c.

```
00027                                      {
00028     s->top++;
00029     if (s->top == s->mem_size) {
00030         void** newptr = realloc(s->base, sizeof(void*) * (s->mem_size + 10));
00031         assert(newptr);
00032         s->base = newptr;
00033         s->mem_size += 10;
00034     }
00035     s->base[s->top] = e;
00036 }
```

4.2 stack.c

```
00001
00009 #include "stack.h"
00010
00017 int stack_isempty(stack_t* s) {
00018     return s->top == -1;
00019 }
00020
00027 void stack_push(stack_t* s, void* e) {
00028     s->top++;
00029     if (s->top == s->mem_size) {
00030         void** newptr = realloc(s->base, sizeof(void*) * (s->mem_size + 10));
00031         assert(newptr);
00032         s->base = newptr;
00033         s->mem_size += 10;
00034     }
00035     s->base[s->top] = e;
00036 }
00037
00044 void* stack_pop(stack_t* s) {
00045     if (stack_isempty(s)) return NULL;
00046     s->top--;
00047     return s->base[s->top + 1];
00048 }
00049
00057 stack_t* stack_create(size_t width) {
00058     stack_t* s = malloc(sizeof(stack_t));
00059     assert(s);
00060     s->width = width;
00061     s->mem_size = 10;
```

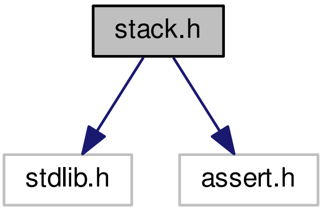
```
    s->base = (void**) malloc(sizeof(void*) * s->mem_size);
    assert(s->base);
    s->top = -1;
    return s;
}

void stack_destruct(stack_t* s) {
    free(s->base);
    free(s);
}
```

4.3 stack.h File Reference

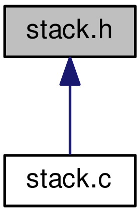
stack definition and basic operations

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

Include dependency graph for stack.h:



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



Data Structures

- struct stack_t

Functions

- stack_t ∗ stack_create (size_t width)
- void stack_destruct (stack_t ∗s)
- int stack_isempty (stack_t ∗s)
- void ∗ stack_pop (stack_t ∗s)
- void stack_push (stack_t ∗s, void ∗e)

4.3.1 Detailed Description

stack definition and basic operations

Author

Firmin MARTIN

Version

0.1

Date

28/12/2017

Definition in file stack.h.

4.3.2 Function Documentation

4.3.2.1 stack_t∗ stack_create (size_t *width*)

Given the size of each element, create a stack 10 ∗ sizeof(void∗) bytes is reserved by default.

Parameters

width	size of each element

Returns

a stack initialized

Definition at line 57 of file stack.c.

```
00057                                            {
00058     stack_t* s = malloc(sizeof(stack_t));
00059     assert(s);
00060     s->width = width;
00061     s->mem_size = 10;
00062     s->base = (void**) malloc(sizeof(void*) * s->mem_size);
00063     assert(s->base);
00064     s->top = -1;
00065     return s;
00066 }
```

4.3.2.2 void stack_destruct (**stack_t** ∗ ***s***)

Free a stack.

Parameters

s	a stack

Definition at line 73 of file stack.c.

```
00073                                  {
00074     free(s->base);
00075     free(s);
00076 }
```

4.3.2.3 int stack_isempty (**stack_t** ∗ ***s***)

Determinate the emptiness of a stack.

Parameters

s	stack

Returns

1 if the stack s is empty, 0 otherwise.

Definition at line 17 of file stack.c.

```
00017                               {
00018     return s->top == -1;
00019 }
```

4.3.2.4 void∗ stack_pop (**stack_t** ∗ ***s***)

Pop out an element from the stack s.

Parameters

s	stack

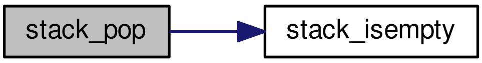
Returns

an element

Definition at line 44 of file stack.c.

```
00044                            {
00045     if (stack_isempty(s)) return NULL;
00046     s->top--;
00047     return s->base[s->top + 1];
00048 }
```

Here is the call graph for this function:



4.3.2.5 void stack_push (stack_t * *s*, void * *e*)

Push an element e into the stack s.

Parameters

s	stack
e	element which be pushed

Definition at line 27 of file stack.c.

```
00027                                       {
00028     s->top++;
00029     if (s->top == s->mem_size) {
00030         void** newptr = realloc(s->base, sizeof(void*) * (s->mem_size + 10));
00031         assert(newptr);
00032         s->base = newptr;
00033         s->mem_size += 10;
00034     }
00035     s->base[s->top] = e;
00036 }
```

4.4 stack.h

```
#ifndef STACK_H
#define STACK_H

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

typedef struct {
    size_t width;
    int top;
    void** base;
    int mem_size;
} stack_t;

stack_t* stack_create(size_t width);
void stack_destruct(stack_t* s);
int stack_isempty(stack_t* s);
void* stack_pop(stack_t* s);
void stack_push(stack_t* s, void* e);

#endif /* ifndef STACK_H */
```

Index