

My Project

Generated by Doxygen 1.9.2

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 Stack Struct Reference	5
3.1.1 Detailed Description	5
4 File Documentation	7
4.1 stack.h File Reference	7
4.1.1 Macro Definition Documentation	8
4.1.1.1 FLOAT_DATA	8
4.1.1.2 STACK_GENERAL_CHECK	8
4.1.1.3 STACK_POP_ERROR_CHECK	9
4.1.1.4 STACK_RESIZE_ERROR_CHECK	9
4.1.2 Typedef Documentation	9
4.1.2.1 canary_t	9
4.1.2.2 data_t	9
4.1.2.3 hash_t	10
4.1.3 Enumeration Type Documentation	10
4.1.3.1 StackErrors	10
4.1.4 Function Documentation	10
4.1.4.1 Hash()	10
4.1.4.2 StackCtor()	11
4.1.4.3 StackCtorCheck()	11
4.1.4.4 StackDestroy()	12
4.1.4.5 StackDtor()	12
4.1.4.6 StackDtorCheck()	12
4.1.4.7 StackDump()	13
4.1.4.8 StackErrorCheck()	13
4.1.4.9 StackHash()	13
4.1.4.10 StackPop()	14
4.1.4.11 StackPush()	14
4.1.4.12 StackResize()	15
4.2 stack.h	15
Index	19

Chapter 1

Class Index

1.1 Class List

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

Stack	5
---------------------------------	---

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

stack.h	7
-----------------------------------	---

Chapter 3

Class Documentation

3.1 Stack Struct Reference

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

Public Attributes

- [canary_t](#) **canary_left**
- int **size**
- int **capacity**
- [data_t](#) * **data**
- [hash_t](#) **hash**
- [canary_t](#) **canary_right**

3.1.1 Detailed Description

Structure that defines the stack: size, capacity, array pointer, hash, canary left and right for protection.

Parameters

in	<i>canary_left</i>	canary_left-left canary for protection
in	<i>size</i>	size-stack size
in	<i>capacity</i>	capacity-stack capacity
in	<i>data</i>	data-pointer to the beginning of the array
in	<i>hash</i>	b-coefficient
in	<i>canary_right</i>	canary_right-left right for protection

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

- [stack.h](#)

Chapter 4

File Documentation

4.1 stack.h File Reference

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

Classes

- struct [Stack](#)

Macros

- #define [FLOAT_DATA](#)
- #define [STACK_GENERAL_CHECK](#)(check_function)
- #define [STACK_RESIZE_ERROR_CHECK](#)()
- #define [STACK_POP_ERROR_CHECK](#)()

Typedefs

- typedef double [data_t](#)
- typedef size_t [canary_t](#)
- typedef size_t [hash_t](#)

Enumerations

- enum [StackErrors](#) {
 STK_IS_NULL_PTR = 1 , DATA_IS_NULL_PTR = 2 , STK_DESTROYED = 4 , STK_OVERFL = 8 ,
 STK_UNDERFL = 16 , STK_DOUBLE_CTED = 32 , STRCT_CANARY_BAD = 64 , DATA_CANARY_BAD
 = 128 ,
 HASH_BAD = 256 , CAPACITY_LARG_SIZE = 512 }

4.1.1.3 STACK_POP_ERROR_CHECK

```
#define STACK_POP_ERROR_CHECK( )
```

Value:

```
do
{
    Errors = 0;

    StackErrorCheck(stk);

    if (stk->size <= 0)
    {
        Errors |= STK_UNDERFL;
    }

    StackDump(stk, __FILE__, __FUNCTION__);

    if (Errors != 0)
    {
        return (data_t) 0xBEDABEDA;
    }
} while (0)
```

Define which describes the general check of the stack during the popping of the element.

4.1.1.4 STACK_RESIZE_ERROR_CHECK

```
#define STACK_RESIZE_ERROR_CHECK( )
```

Value:

```
do
{
    Errors = 0;

    StackErrorCheck(stk);

    StackDump(stk, __FILE__, __FUNCTION__);

    if (Errors != 0)
    {
        return nullptr;
    }
} while (0)
```

Define which describes a general stack check during stack resizing.

4.1.2 Typedef Documentation

4.1.2.1 canary_t

```
typedef size_t canary_t
```

Typedef which specifies that the type of all elements on the stack is int. Typedef which indicates what type of canary is.

4.1.2.2 data_t

```
typedef double data_t
```

Typedef which specifies that the type of all elements on the stack is float.

4.1.2.3 hash_t

```
typedef size_t hash_t
```

Typedef which indicates what type of hash is.

4.1.3 Enumeration Type Documentation

4.1.3.1 StackErrors

```
enum StackErrors
```

Enum which describes all error codes and their numbers.

Parameters

in	<i>STK_IS_NULL_PTR</i>	
in	<i>DATA_IS_NULL_PTR</i>	
in	<i>STK_DESTROYED</i>	
in	<i>STK_OVERFL</i>	
in	<i>STK_UNDERFL</i>	
in	<i>STK_DOUBLE_CTED</i>	
in	<i>STRCT_CANARY_BAD</i>	
in	<i>DATA_CANARY_BAD</i>	
in	<i>HASH_BAD</i>	
in	<i>CAPACITY_LARG_SIZE</i>	

4.1.4 Function Documentation

4.1.4.1 Hash()

```
size_t Hash (  
    void * memory,  
    size_t size_memory )
```

Function that describes the process of hashing a separate piece of memory.

Parameters

in	<i>memory</i>	memory-pointer to memory area
in	<i>size_memory</i>	size_memory-memory area size
out	<i>hash</i>	hash-memory area hash values

Returns

memory area hash values

4.1.4.2 StackCtor()

```
int StackCtor (  
    Stack * stk )
```

Function that describes the stack constructor.

Parameters

in	<i>stk</i>	stk-stack pointer
----	------------	-------------------

Returns

zero

4.1.4.3 StackCtorCheck()

```
int StackCtorCheck (  
    Stack * stk )
```

Function that describes the process of checking the stack while the constructor is running.

Parameters

in	<i>stk</i>	stk-stack pointer
----	------------	-------------------

Returns

zero

4.1.4.4 StackDestroy()

```
int StackDestroy (
    Stack * stk )
```

Function that describes the stack breaking process.

Parameters

in	stk	stk-stack pointer
----	-----	-------------------

Returns

nothing

4.1.4.5 StackDtor()

```
int StackDtor (
    Stack * stk )
```

Function that describes the stack destructor.

Parameters

in	stk	stk-stack pointer
----	-----	-------------------

Returns

zero

4.1.4.6 StackDtorCheck()

```
int StackDtorCheck (
    Stack * stk )
```

Function that describes the process of checking the stack while the destructor is running.

Parameters

in	stk	stk-stack pointer
----	-----	-------------------

Returns

zero

4.1.4.7 StackDump()

```
void StackDump (
    Stack * stk,
    const char * current_file,
    const char * current_function )
```

Function that writes all errors to the Dump.txt file.

Parameters

in	<i>stk</i>	stk-stack pointer
in	<i>current_file-the</i>	file in which the error occurred
in	<i>current_function-the</i>	function in which the error occurred

Returns

nothing

4.1.4.8 StackErrorCheck()

```
int StackErrorCheck (
    Stack * stk )
```

Function that describes the process of checking the stack.

Parameters

in	<i>stk</i>	stk-stack pointer
out	<i>Errors</i>	Error-number of errors in the stack

Returns

number of errors in the stack

4.1.4.9 StackHash()

```
size_t StackHash (
    Stack * stk )
```

Function that describes the stack hashing process.

Parameters

in	<i>stk</i>	stk-stack pointer
out	<i>hash</i>	hash-stack hash value

Returns

stack hash value

4.1.4.10 StackPop()

```
data_t StackPop (
    Stack * stk )
```

Function that describes the process of popping the value of an element from the top of the stack.

Parameters

in	<i>stk</i>	stk-stack pointer
out	<i>data_pop</i>	data_pop-value of the element that is popped from the top of the stack

Returns

value of the element that is popped from the top of the stack

4.1.4.11 StackPush()

```
int StackPush (
    Stack * stk,
    data_t value )
```

Function that describes the process of pushing an element onto the stack.

Parameters

in	<i>stk</i>	stk-stack pointer
in	<i>value</i>	value-value of the element that is pushed onto the stack

Returns

zero

4.1.4.12 StackResize()

```
data_t * StackResize (
    Stack * stk )
```

Function that describes the process of changing the stack size.

Parameters

in	<i>stk</i>	stk-stack pointer
out	<i>new_adress</i>	new_adress-new stack pointer

Returns

new stack pointer

4.2 stack.h

[Go to the documentation of this file.](#)

```
1
2 #pragma once
3
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <string.h>
7 #include <assert.h>
8 #include <limits.h>
9
10 //-----
14 //-----
15
16 #define FLOAT_DATA
17
18 //-----
22 //-----
23
24 #ifdef FLOAT_DATA
25 typedef double data_t;
26 #endif
27
28 //-----
32 //-----
33
34 #ifdef INT_DATA
35 typedef int data_t;
36 #endif
37
38 //-----
42 //-----
43
44 typedef size_t canary_t;
45
46 //-----
50 //-----
51
52 typedef size_t hash_t;
53
54 //-----
64 //-----
65
66 struct Stack
67 {
68     canary_t    canary_left;
69
70     int         size;
71     int         capacity;
72     data_t*     data;
73     hash_t      hash;
74
75     canary_t    canary_right;
76 };
```

```

77
78 //-----
92 //-----
93
94 enum StackErrors
95 {
96     STK_IS_NULL_PTR      = 1,
97     DATA_IS_NULL_PTR   = 2,
98     STK_DESTROYED       = 4,
99     STK_OVERFL          = 8,
100    STK_UNDERFL          = 16,
101    STK_DOUBLE_CTED      = 32,
102    STRCT_CANARY_BAD      = 64,
103    DATA_CANARY_BAD      = 128,
104    HASH_BAD              = 256,
105    CAPACITY_LARG_SIZE    = 512,
106 };
107
108 //enum ResizeTypes
109 //{
110 //    RESIZESMALLER = 0,
111 //    RESIZELARGER = 1
112 //};
113
114 //-----
117 //-----
118
119 #define STACK_GENERAL_CHECK(check_function)
120 do
121 {
122     Errors = 0;
123     check_function;
124     StackDump(stk, __FILE__, __FUNCTION__);
125     if (Errors != 0)
126     {
127         return 1;
128     }
129 } while (0)
130
131 //-----
137 //-----
138
139 #define STACK_RESIZE_ERROR_CHECK()
140 do
141 {
142     Errors = 0;
143     StackErrorCheck(stk);
144     StackDump(stk, __FILE__, __FUNCTION__);
145     if (Errors != 0)
146     {
147         return nullptr;
148     }
149 } while (0)
150
151 //-----
157 //-----
158
159 #define STACK_POP_ERROR_CHECK()
160 do
161 {
162     Errors = 0;
163     StackErrorCheck(stk);
164     if (stk->size <= 0)
165     {
166         Errors |= STK_UNDERFL;
167     }
168     StackDump(stk, __FILE__, __FUNCTION__);
169     if (Errors != 0)
170     {
171         return (data_t) 0xBEDABEDA;
172     }
173 } while (0)
174
175 //-----
186 //-----
187
188 int StackCtor(Stack* stk);

```

```
189
190 //-----
197 //-----
198
199 int StackDtor(Stack* stk);
200
201 //-----
209 //-----
210
211 data_t* StackResize(Stack* stk);
212
213 //-----
221 //-----
222
223 int StackPush(Stack* stk, data_t value);
224
225 //-----
233 //-----
234
235 data_t StackPop(Stack* stk);
236
237 //-----
245 //-----
246
247 int StackErrorCheck(Stack* stk);
248
249 //-----
256 //-----
257
258 int StackCtorCheck (Stack* stk);
259
260 //-----
267 //-----
268
269 int StackDtorCheck (Stack* stk);
270
271 //-----
280 //-----
281
282 void StackDump (Stack* stk, const char* current_file, const char* current_function);
283
284 //-----
292 //-----
293
294 size_t StackHash (Stack* stk);
295
296 //-----
305 //-----
306
307 size_t Hash (void* memory, size_t size_memory);
308
309 //-----
316 //-----
317
318 int StackDestroy(Stack* stk);
```


Index

canary_t
 stack.h, [9](#)

data_t
 stack.h, [9](#)

FLOAT_DATA
 stack.h, [8](#)

Hash
 stack.h, [10](#)

hash_t
 stack.h, [9](#)

Stack, [5](#)

stack.h, [7](#)

 canary_t, [9](#)

 data_t, [9](#)

 FLOAT_DATA, [8](#)

 Hash, [10](#)

 hash_t, [9](#)

 STACK_GENERAL_CHECK, [8](#)

 STACK_POP_ERROR_CHECK, [8](#)

 STACK_RESIZE_ERROR_CHECK, [9](#)

 StackCtor, [11](#)

 StackCtorCheck, [11](#)

 StackDestroy, [11](#)

 StackDtor, [12](#)

 StackDtorCheck, [12](#)

 StackDump, [13](#)

 StackErrorCheck, [13](#)

 StackErrors, [10](#)

 StackHash, [13](#)

 StackPop, [14](#)

 StackPush, [14](#)

 StackResize, [14](#)

STACK_GENERAL_CHECK
 stack.h, [8](#)

STACK_POP_ERROR_CHECK
 stack.h, [8](#)

STACK_RESIZE_ERROR_CHECK
 stack.h, [9](#)

StackCtor
 stack.h, [11](#)

StackCtorCheck
 stack.h, [11](#)

StackDestroy
 stack.h, [11](#)

StackDtor
 stack.h, [12](#)

StackDtorCheck
 stack.h, [12](#)

StackDump
 stack.h, [13](#)

StackErrorCheck
 stack.h, [13](#)

StackErrors
 stack.h, [10](#)

StackHash
 stack.h, [13](#)

StackPop
 stack.h, [14](#)

StackPush
 stack.h, [14](#)

StackResize
 stack.h, [14](#)