salibc

Generated by Doxygen 1.8.11

Contents

3 Data Structure Documentation

3.1 Array Struct Reference

Array Abstract Data Type.
`#include <salibc.h>`

Data Fields

- size_t size

  Size of a single element.
- int nmemb

  Number of elements contained in the array.
- char * ptr

  Pointer to the array.

3.1.1 Detailed Description

Array Abstract Data Type.

3.1.2 Field Documentation

3.1.2.1 char* Array::ptr

Pointer to the array.
Since pointer arithmetic cannot be done on void *, char * was the obvious choice.

3.1.2.2 size_t Array::size

Size of a single element.
This is expressed in bytes.
The documentation for this struct was generated from the following file:

- salibc.h

4 File Documentation

4.1 salibc.h File Reference

Header file containing exportable methods.

```
#include <assert.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Data Structures

- struct Array

  Array Abstract Data Type.

Macros

- #define SALIBC

  Include the main header.
- #define ISOC99_SOURCE

  Tell the compiler that we want ISO C99 source, and check if the system has ANSI C 99.

Typedefs

- typedef struct Array * **Array**

Functions

- bool array_null (Array a)

  Check if the array is NULL.
- bool array_empty (Array a)

  Check if the array is empty.
- size_t array_size (Array a)

  Get the size in bytes of a single element of the array.
- int array_length (Array a)

  Get the number of elements contained in the array.
- size_t array_fullsize (Array a)

  Get the size in bytes of all the elements of the array.
- char * array_pointer (Array a)

  Get the memory address of the first element of the array.
- bool array_equal (Array a1, Array a2)

  Check if two arrays are equal.
- void array_delete (Array *a_ref)

  Delete the ADT instance of the array.
- Array array_new (int nmemb, size_t size)

  Create a new array ADT instance. This is also known as the constructor.
- bool array_put (Array a, int index, void *element)

  Insert an element into an array ADT instance.
- bool array_set (Array a, void *element)

  Set the whole array with the same element.
- char * array_get (Array a, int index)

  Get the memory address corresponding to a specified index of the array.

- Array array_copy (Array a1)

  Get a copy of the specified array ADT.
- bool array_resize (Array a, int new_length)

  Resize an array to a new specified length.
- bool array_append (Array a, void ∗element)

  Append (add on the tail) a new element on the array.
- char ∗ array_trim (Array a)

  Get the last element of the array and remove the last position from it .
- Array array_merge (Array a1, Array a2)

  Merge two arrays in a new array.

4.1.1 Detailed Description

Header file containing exportable methods.

Author

Franco Masotti

Date

28 Apr 2016

4.1.2 Function Documentation

4.1.2.1 bool array_append (Array *a,* void ∗ *element*)

Append (add on the tail) a new element on the array.

Parameters

in	*a*	The pointer to an array ADT instance.
in	*element*	A memory address of the element to be inserted.

Return values

true	Array append successful.
false	Array append unsuccessful.

Note

This function alters the input.

4.1.2.2 Array array_copy (Array *a1*)

Get a copy of the specified array ADT.

Parameters

in	*a1*	The pointer to an array ADT instance.

Return values

a2	The pointer to the new array ADT istance.

Warning

This function may return NULL if some problem occured.

Note

Allocate a new array with the same ADT characteristics.

```
1  */
2 a2 = array_new (array_length (a1), array_size (a1));
3 if (array_null (a2))
4   return NULL;
5 /**
```

Note

Copy the real array using the previously defined functions.

```
1  */
2 for (i = 0; i < array_length (a1); i++)
3   if (!array_memcopy (a2, i, array_indexpointer (a1, i)))
4     return NULL;
5 /**
```

4.1.2.3 void array_delete (Array * *a_ref*)

Delete the ADT instance of the array.

Parameters

in	*a_ref*	The memory address of the variable containing the pointer to the array ADT instance.

4.1.2.4 bool array_empty (Array *a*)

Check if the array is empty.

Parameters

in	*a*	The pointer to an array ADT instance.

Return values

true	The array is empty.
false	The array is not empty.

When an array is empty, it means that it does not contain any element (i.e: its length is zero.

4.1.2.5 bool array_equal (Array *a1*, Array *a2*)

Check if two arrays are equal.

Parameters

in	*a1*	The pointer to the first array ADT instance.
in	*a2*	The pointer to the second array ADT instance.

Return values

true	The two arrays are equal.
false	The two arrays differ.

Note

memcmp works well in checking equality even for floating point numbers.

4.1.2.6 size_t array_fullsize (Array *a*)

Get the size in bytes of all the elements of the array.

Parameters

in	*a*	The pointer to an array ADT instance.

Return values

array_size(a)∗*array_length(a)*	The total size in bytes of the array.

Precondition

a must not be NULL.

Note

This function should not return an out of bound value.

4.1.2.7 char∗ array_get (Array *a,* int *index*)

Get the memory address corresponding to a specified index of the array.

Parameters

in	*a*	The pointer to an array ADT instance.
in	*index*	The index of the array where to get the element.

Return values

array_indexpointer()	A memory address corresponding to the input index.

Warning

This function may return NULL if some problem occured.

Note

If you dereference the return value with the correct pointer type you get the real value value that can be used in arthmetics and printing.
This function is an interface to array_indexpointer.

4.1.2.8 int array_length (Array *a*)

Get the number of elements contained in the array.

Parameters

in	*a*	The pointer to an array ADT instance.

Return values

a->nmemb	The length of the array.

Precondition

a must not be NULL.

4.1.2.9 Array array_merge (Array *a1*, Array *a2*)

Merge two arrays in a new array.

Parameters

in	*a1*	The pointer the first array ADT instance.
in	*a2*	The pointer the second array ADT instance.

Return values

a2	The pointer to the new array ADT istance.

Warning

This function may return NULL if some problem occured.

Note

Safety controls.

```
1  */
2 if ((array_null (a1) && array_null (a2)) || (array_size (a1) !=
3                       array_size (a2)))
4   return NULL;
5 /**
```

4.1.2.10 Array array_new (int *nmemb*, size_t *size*)

Create a new array ADT instance. This is also known as the constructor.

Parameters

in	*nmemb*	The length of the array.
in	*size*	The size of each element, in bytes.

Return values

new_array	A pointer to the new array ADT instance.

Warning

The return value can also be NULL if some problem occurred.

Note

This function is also known as the array constructor.

4.1.2.11 bool array_null (Array *a*)

Check if the array is NULL.

Parameters

in	*a*	The pointer to an array ADT instance.

Return values

true	The array is NULL.
false	The array is not NULL.

4.1.2.12 char* array_pointer (Array *a*)

Get the memory address of the first element of the array.

Parameters

in	*a*	The pointer to an array ADT instance.

Return values

a->ptr	The pointer to the first element of the array.

4.1.2.13 bool array_put (Array *a*, int *index*, void * *element*)

Insert an element into an array ADT instance.

Parameters

in	*a*	The pointer to an array ADT instance.
in	*index*	The index of the array where to store the element.
in	*element*	A memory address of the element to be inserted.

Return values

true	The element has been inserted correctly.
false	Some problem occurred and insertion failed.

4.1.2.14 bool array_resize (Array *a*, int *new_length*)

Resize an array to a new specified length.

Parameters

in	*a*	The pointer to an array ADT instance.
in	*new_length*	The new length of the array.

Return values

true	Array resize successful.
false	Array resize unsuccessful.

```
1  */
2 /*
```

```
3  * Invalid new length.
4  */
5 if (new_length < 0)
6   return false;
7 /*
8  * new_length is set to zero, so leave the ADT, but delete internal array.
9  */
10 else if (new_length == 0)
11   {
12     realarray_delete (a);
13     return true;
14   }
15 /*
16  * Same size as before, then do nothing.
17  */
18 else if (array_length (a) == new_length)
19   return true;
20 /*
21  * Array's length != new_length, so realloc can now be used directly.
22  */
23 else
24   {
25     /*
26      * Safe realloc (to avoid losing the stored array if realloc fails).
27      */
28     tmp =
29   realloc (array_pointer (a),
30      array_fullsize (a) +
31      (array_size (a) * ((size_t) new_length)));
32     if (!element_null (tmp))
33   a->ptr = tmp;
34     else
35   return false;
36     /*
37      * memset to 0 new part of the array.
38      * To do this we must go to the first byte of the new array and put 0
39      * until we get to (memdiff * a->size) bytes.
40      */
41     memdiff = new_length - array_length (a);
42     if (memdiff > 0)
43   memset (array_pointer (a) + array_fullsize (a) + array_size (a), 0,
44     ((size_t) memdiff) * array_size (a));
45
46     /*
47      * Set the new array length.
48      */
49     a->nmemb = new_length;
50   }
51
52 return true;
53 /**
```

4.1.2.15 bool array_set (Array *a*, void * *element*)

Set the whole array with the same element.

Parameters

in	*a*	The pointer to an array ADT instance.
in	*element*	A memory address of the element to be inserted.

Return values

true	The entire array has been set correctly.
false	Some problem occurred and insertion in one of the array's index failed.

Warning

This function may leave an undefined state of the array.

4.1.2.16 size_t array_size (Array *a*)

Get the size in bytes of a single element of the array.

Parameters

in	*a*	The pointer to an array ADT instance.

Return values

a->size	The size of the array.

Precondition

a must not be NULL.

4.1.2.17 char* array_trim (Array *a*)

Get the last element of the array and remove the last position from it .

Parameters

in	*a*	The pointer to an array ADT instance.

Return values

element_copy	A pointer to the value that was in the last array index.

Warning

The return value can also be NULL if some problem occurred.

Note

Copy *element int *element_copy.

```
1  */
2 element = array_indexpointer (a, initial_length - 1);
3 element_copy = malloc (array_size (a));
4 memcpy (element_copy, element, array_size (a));
5 /**
```

4.2 salibc_test.c File Reference

Test file.

```
#include "salibc.h"
```

Functions

- int main (void)

  if this flag is defined then the main function in this file is included.

4.2.1 Detailed Description

Test file.

Author

Franco Masotti

Date

28 Apr 2016

4.2.2 Function Documentation

4.2.2.1 int main (void)

if this flag is defined then the main function in this file is included.

Note

Use: const MYVARIABLE = value instead of: value or #define MYVAR value in your code.