

C-Array

0.1.3

Generated by Doxygen 1.8.11

Contents

1	carray	1
2	Module Index	3
2.1	Modules	3
3	Data Structure Index	5
3.1	Data Structures	5
4	File Index	7
4.1	File List	7
5	Module Documentation	9
5.1	C-Array class	9
5.1.1	Detailed Description	10
5.1.2	Function Documentation	10
5.1.2.1	carray_add(carray *c, int index, type new_value, void **ok)	10
5.1.2.2	carray_addspace(carray *c, void **ok)	11
5.1.2.3	carray_adjust(carray *c, void **ok)	11
5.1.2.4	carray_append(carray *c, carray *o, void **ok)	11
5.1.2.5	carray_clear(carray *c)	12
5.1.2.6	carray_concat_TF(carray *a, carray *b)	12
5.1.2.7	carray_contains(carray *c, type test_element, bool(*eqfunc)(type, type))	12
5.1.2.8	carray_equal(carray *a, carray *b, bool(*eqfunc)(type, type))	12
5.1.2.9	carray_free(carray *c, void(*voidfunc)(type))	13
5.1.2.10	carray_free_obj(type val)	13

5.1.2.11	<code>carray_get(carray *c, int index, void **ok)</code>	13
5.1.2.12	<code>carray_getarray(carray *c)</code>	14
5.1.2.13	<code>carray_getreadposition(carray *c)</code>	14
5.1.2.14	<code>carray_getsize(carray *c)</code>	14
5.1.2.15	<code>carray_getspace(carray *c)</code>	15
5.1.2.16	<code>carray_hashcode(carray *c, hashtype(*hashfunc)(type))</code>	15
5.1.2.17	<code>carray_indexof(carray *c, type test_value, bool(*eqfunc)(type, type))</code>	15
5.1.2.18	<code>carray_ins(carray *c, type value)</code>	16
5.1.2.19	<code>carray_isempty(carray *c)</code>	16
5.1.2.20	<code>carray_lastindexof(carray *c, type test_value, bool(*eqfunc)(type, type))</code>	16
5.1.2.21	<code>carray_new()</code>	16
5.1.2.22	<code>carray_new_CC(carray *copy_carray)</code>	17
5.1.2.23	<code>carray_new_CISC(carray *copy_carray, size_t init_space)</code>	17
5.1.2.24	<code>carray_new_ISC(size_t init_space)</code>	17
5.1.2.25	<code>carray_pop(carray *c)</code>	18
5.1.2.26	<code>carray_push(carray *c, type value)</code>	18
5.1.2.27	<code>carray_read(carray *c, void **ok)</code>	18
5.1.2.28	<code>carray_readingsremaining(carray *c)</code>	19
5.1.2.29	<code>carray_remove(carray *c, int index, void **ok)</code>	19
5.1.2.30	<code>carray_remove_elt(carray *c, type test_element, bool(*eqfunc)(type, type))</code>	19
5.1.2.31	<code>carray_reverse(carray *c)</code>	20
5.1.2.32	<code>carray_reversed_TF(carray *c)</code>	20
5.1.2.33	<code>carray_safeset(carray *c, int index, type value, void **ok)</code>	20
5.1.2.34	<code>carray_set(carray *c, int index, type new_value, void **ok)</code>	20
5.1.2.35	<code>carray_setreadposition(carray *c, int new_read_position, void **ok)</code>	21
5.1.2.36	<code>carray_setspace(carray *c, size_t new_space, void **ok)</code>	21
5.1.2.37	<code>carray_subarray_TF(carray *c, int from_index, int to_index, void **ok)</code>	21
5.1.2.38	<code>carray_subarraystep_TF(carray *c, int from_index, int to_index, int step, void **ok)</code>	22
5.1.2.39	<code>carray_subcarray_TF(carray *c, int from_index, int to_index, void **ok)</code>	22
5.1.2.40	<code>carray_subcarraystep_TF(carray *c, int from_index, int to_index, int step, void **ok)</code>	22

5.1.2.41	<code>carray_toarray_TF(carray *c)</code>	23
5.1.2.42	<code>carray_tostring_TF(carray *c, char *(*strfunc)(type), char *opener, char *closer, char *prefix, char *suffix)</code>	23
5.1.2.43	<code>fatal_error(char message[])</code>	24
5.2	Constants for C-Array class	25
5.2.1	Detailed Description	25
5.2.2	Macro Definition Documentation	25
5.2.2.1	<code>DEFAULT_SHRINK_PERCENT</code>	25
5.2.2.2	<code>DEFAULT_SHRINK_THRESHOLD</code>	25
5.2.2.3	<code>DEFAULT_SPACE_INCR</code>	25
5.2.2.4	<code>DEFAULT_SPACE_INIT_FLAT</code>	26
5.2.2.5	<code>DEFAULT_SPACE_INIT_PERCENT</code>	26
5.2.3	Typedef Documentation	26
5.2.3.1	<code>carray</code>	26
5.3	Implementation specific constants and aliases	27
5.3.1	Detailed Description	27
5.4	Preprocessor macros	28
5.4.1	Detailed Description	29
5.4.2	Macro Definition Documentation	29
5.4.2.1	<code>Bool</code>	29
5.4.2.2	<code>Char</code>	29
5.4.2.3	<code>Double</code>	30
5.4.2.4	<code>Float</code>	30
5.4.2.5	<code>Int</code>	30
5.4.2.6	<code>IDouble</code>	30
5.4.2.7	<code>ILong</code>	30
5.4.2.8	<code>Long</code>	31
5.4.2.9	<code>sChar</code>	31
5.4.2.10	<code>Short</code>	31
5.4.2.11	<code>sInt</code>	31
5.4.2.12	<code>slong</code>	31

5.4.2.13	sLong	32
5.4.2.14	sShort	32
5.4.2.15	String	32
5.4.2.16	uChar	32
5.4.2.17	ulnt	32
5.4.2.18	ulLong	33
5.4.2.19	uLong	33
5.4.2.20	uShort	33
5.4.3	Function Documentation	33
5.4.3.1	carray_free_obj(type)	33
5.5	C-Array core	34
5.5.1	Detailed Description	34
5.6	C-Array methods	35
5.6.1	Detailed Description	36
5.6.2	Function Documentation	36
5.6.2.1	carray_add(carray *, int, type, void **)	36
5.6.2.2	carray_adjust(carray *, void **)	36
5.6.2.3	carray_append(carray *, carray *, void **)	36
5.6.2.4	carray_clear(carray *)	37
5.6.2.5	carray_concat_TF(carray *, carray *)	37
5.6.2.6	carray_get(carray *, int, void **)	37
5.6.2.7	carray_getarray(carray *)	37
5.6.2.8	carray_getreadposition(carray *)	38
5.6.2.9	carray_getsize(carray *)	38
5.6.2.10	carray_getspace(carray *)	38
5.6.2.11	carray_ins(carray *, type)	39
5.6.2.12	carray_isempty(carray *)	39
5.6.2.13	carray_new()	39
5.6.2.14	carray_new_CC(carray *)	39
5.6.2.15	carray_new_CISC(carray *, size_t)	40

5.6.2.16	<code>carray_new_ISC(size_t)</code>	40
5.6.2.17	<code>carray_pop(carray *)</code>	40
5.6.2.18	<code>carray_push(carray *, type)</code>	41
5.6.2.19	<code>carray_read(carray *, void **ok)</code>	41
5.6.2.20	<code>carray_readingsremaining(carray *)</code>	41
5.6.2.21	<code>carray_remove(carray *, int, void **)</code>	42
5.6.2.22	<code>carray_reverse(carray *)</code>	42
5.6.2.23	<code>carray_reversed_TF(carray *)</code>	42
5.6.2.24	<code>carray_safeset(carray *, int, type, void **)</code>	43
5.6.2.25	<code>carray_set(carray *, int, type, void **)</code>	43
5.6.2.26	<code>carray_setreadposition(carray *, int, void **)</code>	43
5.6.2.27	<code>carray_setspace(carray *, size_t, void **)</code>	43
5.6.2.28	<code>carray_subarray_TF(carray *, int, int, void **)</code>	44
5.6.2.29	<code>carray_subarraystep_TF(carray *, int, int, int, void **)</code>	44
5.6.2.30	<code>carray_subcarray_TF(carray *, int, int, void **)</code>	45
5.6.2.31	<code>carray_subcarraystep_TF(carray *, int, int, int, void **)</code>	45
5.6.2.32	<code>carray_toarray_TF(carray *)</code>	45
6	Data Structure Documentation	47
6.1	<code>carray</code> Struct Reference	47
6.1.1	Detailed Description	47
7	File Documentation	49
7.1	<code>carray.c</code> File Reference	49
7.1.1	Detailed Description	50
7.2	<code>carray.h</code> File Reference	50
7.2.1	Detailed Description	54
	Index	55

Chapter 1

carray

Python-like list for C

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

C-Array class	9
Constants for C-Array class	25
Implementation specific constants and aliases	27
Preprocessor macros	28
C-Array core	34
C-Array methods	35

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

carray	47
-------------------------	----

Chapter 4

File Index

4.1 File List

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

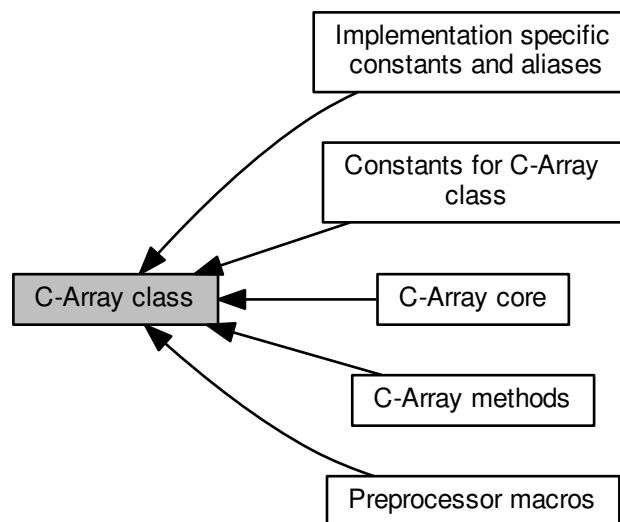
carray.c	Implementation of the carray class. Contains all the function implementations and documentation	49
carray.h	Header of the carray class. Contains all function declarations and preprocessor directives . . .	50
macro_generator.py	??
main.c	??
cmake-build-debug/CMakeFiles/ feature_tests.c	??
cmake-build-debug/CMakeFiles/ feature_tests.cxx	??
cmake-build-debug/CMakeFiles/3.8.2/CompilerIdC/ CMakeCCompilerId.c	??
cmake-build-debug/CMakeFiles/3.8.2/CompilerIdCXX/ CMakeCXXCompilerId.cpp	??

Chapter 5

Module Documentation

5.1 C-Array class

Collaboration diagram for C-Array class:



Modules

- **Constants for C-Array class**
- **Implementation specific constants and aliases**
- **Preprocessor macros**
- **C-Array core**
- **C-Array methods**

Functions

- void **fatal_error** (char message[])
- **carray** * **carray_new** ()
- **carray** * **carray_new_ISC** (size_t init_space)
- **carray** * **carray_new_CC** (**carray** *copy_carray)
- **carray** * **carray_new_CISC** (**carray** *copy_carray, size_t init_space)
- void **carray_free** (**carray** *c, void(*voidfunc)(type))
- size_t **carray_getsize** (**carray** *c)
- size_t **carray_getspace** (**carray** *c)
- void **carray_setspace** (**carray** *c, size_t new_space, void **ok)
- void **carray_addspace** (**carray** *c, void **ok)
- void **carray_safeset** (**carray** *c, int index, type value, void **ok)
- void **carray_append** (**carray** *c, **carray** *o, void **ok)
- size_t **carray_getreadposition** (**carray** *c)
- void **carray_setreadposition** (**carray** *c, int new_read_position, void **ok)
- size_t **carray_readingsremaining** (**carray** *c)
- type * **carray_getarray** (**carray** *c)
- type **carray_read** (**carray** *c, void **ok)
- void **carray_push** (**carray** *c, type value)
- void **carray_ins** (**carray** *c, type value)
- type **carray_pop** (**carray** *c)
- void **carray_adjust** (**carray** *c, void **ok)
- void **carray_reverse** (**carray** *c)
- **carray** * **carray_reversed_TF** (**carray** *c)
- **carray** * **carray_concat_TF** (**carray** *a, **carray** *b)
- hashtype **carray_hashcode** (**carray** *c, hashtype(*hashfunc)(type))
- bool **carray_equal** (**carray** *a, **carray** *b, bool(*eqfunc)(type, type))
- char * **carray_tostring_TF** (**carray** *c, char *(*strfunc)(type), char *opener, char *closer, char *prefix, char *suffix)
- bool **carray_isempty** (**carray** *c)
- bool **carray_contains** (**carray** *c, type test_element, bool(*eqfunc)(type, type))
- type * **carray_toarray_TF** (**carray** *c)
- bool **carray_remove_elt** (**carray** *c, type test_element, bool(*eqfunc)(type, type))
- void **carray_clear** (**carray** *c)
- type **carray_get** (**carray** *c, int index, void **ok)
- void **carray_set** (**carray** *c, int index, type new_value, void **ok)
- void **carray_add** (**carray** *c, int index, type new_value, void **ok)
- type **carray_remove** (**carray** *c, int index, void **ok)
- int **carray_indexof** (**carray** *c, type test_value, bool(*eqfunc)(type, type))
- int **carray_lastindexof** (**carray** *c, type test_value, bool(*eqfunc)(type, type))
- **carray** * **carray_subcarray_TF** (**carray** *c, int from_index, int to_index, void **ok)
- **carray** * **carray_subcarraystep_TF** (**carray** *c, int from_index, int to_index, int step, void **ok)
- type * **carray_subarray_TF** (**carray** *c, int from_index, int to_index, void **ok)
- type * **carray_subarraystep_TF** (**carray** *c, int from_index, int to_index, int step, void **ok)
- void **carray_free_obj** (type val)

5.1.1 Detailed Description

5.1.2 Function Documentation

5.1.2.1 void carray_add (carray * c, int index, type new_value, void ** ok)

Adds the specified element at the specified index in the carray. If params are correct and if the operation works, ok will hold the carray address; otherwise, it will hold NULL.

Parameters

<i>c</i>	the carray
<i>index</i>	index where to insert the specified element. The element will be inserted at this exact position, and subsequent elements will be shifted.
<i>new_value</i>	element to insert at the specified index
<i>ok</i>	validation flag

Definition at line 725 of file carray.c.

5.1.2.2 void carray_addspace (carray * *c*, void ** *ok*)

Increases space allowed for the internal representation of the carray. If params are correct and if the operation works, *ok* will hold the carray address; otherwise, it will hold NULL.

Parameters

<i>c</i>	the carray
<i>ok</i>	validation flag

Definition at line 189 of file carray.c.

5.1.2.3 void carray_adjust (carray * *c*, void ** *ok*)

Eventually shrinks the size of the internal representation of the carray if free space exceed the threshold. *ok* will hold the carray address if a shrinking was made, and NULL otherwise.

Parameters

<i>c</i>	the carray
<i>ok</i>	validation flag

Definition at line 386 of file carray.c.

5.1.2.4 void carray_append (carray * *c*, carray * *o*, void ** *ok*)

Append the second specified carray at the end of the first one. If params are correct and if the operation works, *ok* will hold the carray address; otherwise, it will hold NULL.

Parameters

<i>c</i>	the carray
<i>o</i>	the carray to put at the end of the first one
<i>ok</i>	validation flag

Definition at line 251 of file carray.c.

5.1.2.5 void `carray_clear` (`carray * c`)

Clears the content of the carray. Be careful, this function DOESN'T free the space allowed for the internal representation of the carray. Use `carray_adjust` for that.

Parameters

<i>c</i>	the carray
----------	------------

Definition at line 652 of file `carray.c`.

5.1.2.6 `carray*` `carray_concat_TF` (`carray * a`, `carray * b`)

Concatenates both specified carrays into a new one which must be freed after use.

Parameters

<i>a</i>	the first carray to concatenate
<i>b</i>	the second carray to concatenate

Returns

a concatenated version of carrays *a* and *b*

Definition at line 438 of file `carray.c`.

5.1.2.7 bool `carray_contains` (`carray * c`, `type test_element`, `bool(*) (type, type) eqfunc`)

Tests if the carray contains the `test_element` according to the specified equality function

Parameters

<i>c</i>	the carray
<i>test_element</i>	the element to be compared to the carray elements
<i>eqfunc</i>	the function to test equality between elements

Returns

true if the specified element is inside the carray, and false otherwise

Definition at line 598 of file `carray.c`.

5.1.2.8 bool `carray_equal` (`carray * a`, `carray * b`, `bool(*) (type, type) eqfunc`)

Returns true if both carrays are equal and false otherwise, according to the specified elements equality function.

Parameters

<i>a</i>	the first carray to test
<i>b</i>	the second carray to test
<i>eqfunc</i>	equality function to apply on two elements

Returns

true if both carrays are equal, false otherwise

Definition at line 473 of file carray.c.

5.1.2.9 void carray_free (carray * *c*, void(*)(type) *voidfunc*)

Destructor for carray. Frees the carray internal array representation and the carray itself. If *voidfunc* is not NULL, applies this function to each element of the carray before freeing the whole struct.

Parameters

<i>c</i>	the carray
<i>voidfunc</i>	a function to be applied on each element of the carray to free it; can be NULL

Definition at line 120 of file carray.c.

5.1.2.10 void carray_free_obj (type *val*)

Frees the specified primitive-wrapped element.

Parameters

<i>val</i>	primitive-wrapped element to free
------------	-----------------------------------

Definition at line 1077 of file carray.c.

5.1.2.11 type carray_get (carray * *c*, int *index*, void ** *ok*)

Gets the element at the specified index in the carray. If params are correct and if the operation works, *ok* will hold the carray address; otherwise, it will hold NULL.

Parameters

<i>c</i>	the carray
<i>index</i>	index where element will be retrieved
<i>ok</i>	validation flag

Returns

the element at the specified index if index is correct; default type value otherwise

Definition at line 667 of file carray.c.

5.1.2.12 type* carray_getarray (carray * c)

Internal array getter.

Parameters

c	the carray
---	------------

Returns

the internal representation of the carray

Definition at line 320 of file carray.c.

5.1.2.13 size_t carray_getreadposition (carray * c)

Read position getter.

Parameters

c	the carray
---	------------

Returns

the current read position

Definition at line 276 of file carray.c.

5.1.2.14 size_t carray_getsize (carray * c)

Size getter.

Parameters

c	the carray
---	------------

Returns

the size of the carray

Definition at line 138 of file carray.c.

5.1.2.15 `size_t carray_getspace (carray * c)`

Space getter.

Parameters

<i>c</i>	the carray
----------	------------

Returns

the space (in nb of elements) of the carray

Definition at line 148 of file carray.c.

5.1.2.16 `hashtype carray_hashcode (carray * c, hashtype(*) (type) hashfunc)`

Returns the hashcode of the carray according to the specified elements hash code function.

Parameters

<i>c</i>	the carray
<i>hashfunc</i>	hashcode function to apply on each element of the carray

Returns

the hashcode of the carray

Definition at line 455 of file carray.c.

5.1.2.17 `int carray_indexof (carray * c, type test_value, bool(*) (type, type) eqfunc)`

Returns the index of the specified element in the carray, and -1 if this element wasn't found. The carray is read left to right and this function returns the index of the first occurrence met.

Parameters

<i>c</i>	the carray
<i>test_value</i>	the element to find in the carray
<i>eqfunc</i>	the function to test equality between elements

Returns

the index of the specified element if this one was found, -1 otherwise

Definition at line 813 of file carray.c.

5.1.2.18 void `carray_ins` (`carray * c`, type *value*)

Inserts the specified value at the beginning of the carray.

Parameters

<i>c</i>	the carray
<i>value</i>	the element to insert at the end of the carray

Definition at line 362 of file `carray.c`.

5.1.2.19 bool `carray_isempty` (`carray * c`)

Returns true if the carray is empty, false otherwise

Parameters

<i>c</i>	the carray
----------	------------

Returns

true if the carray is empty, false otherwise

Definition at line 584 of file `carray.c`.

5.1.2.20 int `carray_lastindexof` (`carray * c`, type *test_value*, bool(*) (type, type) *eqfunc*)

Returns the index of the specified element in the carray, and -1 if this element wasn't found. The carray is read right to left and this function returns the index of the first occurrence met.

Parameters

<i>c</i>	the carray
<i>test_value</i>	the element to find in the carray
<i>eqfunc</i>	the function to test equality between elements

Returns

the index of the specified element if this one was found, -1 otherwise

Definition at line 838 of file `carray.c`.

5.1.2.21 `carray*` `carray_new` ()

Constructor for carray. Returns a pointer to the created carray which must be freed after use.

Returns

a pointer to the created carray

Definition at line 37 of file carray.c.

5.1.2.22 `carray* carray_new_CC (carray * copy_carray)`

Copy constructor for carray. The created carray is the exact copy of the specified one with a bit more space than the length of the specified one. Returns a pointer to the created carray which must be freed after use.

Parameters

<i>copy_carray</i>	the carray to be copied
--------------------	-------------------------

Returns

a pointer to the created carray

Definition at line 76 of file carray.c.

5.1.2.23 `carray* carray_new_CISC (carray * copy_carray, size_t init_space)`

Copy constructor for carray with `init_space` specified. This carray have at least `init_space` slots at instanciation, and hold the exact content of the specified carray. Returns a pointer to the created carray which must be freed after use; if `init_space` is shorter than the length of the carray to be copied, returns NULL.

Parameters

<i>copy_carray</i>	the carray to be copied
<i>init_space</i>	Initial space (number of elements) of the carray

Returns

a pointer to the created carray

Definition at line 97 of file carray.c.

5.1.2.24 `carray* carray_new_ISC (size_t init_space)`

Constructor for carray with specified `init_space`. This carray have at least `init_space` slots at instanciation. Returns a pointer to the created carray which must be freed after use; if `init_space` is not valid, returns NULL.

Parameters

<i>init_space</i>	Initial space (number of elements) of the carray
-------------------	--

Returns

a pointer to the created carray

Definition at line 55 of file carray.c.

5.1.2.25 type carray_pop (carray * c)

Removes the last element of the carray and returns it.

Parameters

<i>c</i>	the carray
----------	------------

Returns

the last element of the carray

Definition at line 373 of file carray.c.

5.1.2.26 void carray_push (carray * c, type value)

Pushes the specified value at the end of the carray.

Parameters

<i>c</i>	the carray
<i>value</i>	the element to push at the end of the carray

Definition at line 351 of file carray.c.

5.1.2.27 type carray_read (carray * c, void ** ok)

Read method. Returns elt at the current read position and increases read position by 1. If params are correct and if the operation works, ok will hold the carray address; otherwise, it will hold NULL.

Parameters

<i>c</i>	the carray
<i>ok</i>	validation flag

Returns

the element at the current read position; default type value if there's no element to be read

Definition at line 335 of file carray.c.

5.1.2.28 `size_t array_readingsremaining (array * c)`

Gets the number of read operation remaining before reaching the end of the carray.

Parameters

<i>c</i>	the carray
----------	------------

Returns

the number of read possible before the end of the carray

Definition at line 310 of file array.c.

5.1.2.29 `type array_remove (array * c, int index, void ** ok)`

Removes the element at the specified index and returns it. If params are correct and if the operation works, ok will hold the carray address; otherwise, it will hold NULL.

Parameters

<i>c</i>	the carray
<i>index</i>	index where element will be removed. The element will be removed at this exact position, and subsequent elements will be shifted.
<i>ok</i>	validation flag

Returns

the element which have been removed if the index is correct, default type value otherwise.

Definition at line 778 of file array.c.

5.1.2.30 `bool array_remove_elt (array * c, type test_element, bool (*)(type, type) eqfunc)`

Removes the specified element from the carray if and only if this one is present.

Parameters

<i>c</i>	the carray
<i>test_element</i>	element to be removed if present
<i>eqfunc</i>	the function to test equality between elements

Returns

true if the specified element was in the carray and thus was removed, false if nothing was made.

Definition at line 634 of file array.c.

5.1.2.31 void carray_reverse (carray * c)

Reverses the specified carray in-place.

Parameters

<i>c</i>	the carray
----------	------------

Definition at line 402 of file carray.c.

5.1.2.32 carray* carray_reversed_TF (carray * c)

Returns a reversed copy of the specified carray which must be freed after use.

Parameters

<i>c</i>	the carray
----------	------------

Returns

a reversed copy of the specified carray

Definition at line 419 of file carray.c.

5.1.2.33 void carray_safeset (carray * c, int *index*, type *value*, void ** *ok*)

Used to set an element somewhere in the carray even if this cell is not already used by the carray, for example, to set carray[8] with a length-4 carray. Unused cells until the specified one are set with the default type value. If params are correct and if the operation works, ok will hold the carray address; otherwise, it will hold NULL.

Parameters

<i>c</i>	the carray
<i>index</i>	index where to put the specified value; can be larger than the carray size
<i>value</i>	value to be put at the specified index
<i>ok</i>	validation flag

Definition at line 207 of file carray.c.

5.1.2.34 void carray_set (carray * c, int *index*, type *new_value*, void ** *ok*)

Sets the element at the specified index in the carray to the specified value. If params are correct and if the operation works, ok will hold the carray address; otherwise, it will hold NULL.

Parameters

<i>c</i>	the carray
----------	------------

Parameters

<i>index</i>	index where element will be set
<i>new_value</i>	new value for this element
<i>ok</i>	validation flag

Definition at line 697 of file `carray.c`.

5.1.2.35 void `carray_setreadposition` (`carray * c`, `int new_read_position`, `void ** ok`)

Read position setter. If params are correct and if the operation works, `ok` will hold the `carray` address; otherwise, it will hold `NULL`.

Parameters

<i>c</i>	the <code>carray</code>
<i>new_read_position</i>	new read position
<i>ok</i>	validation flag

Definition at line 289 of file `carray.c`.

5.1.2.36 void `carray_setspace` (`carray * c`, `size_t new_space`, `void ** ok`)

Space setter. Can be used to modify the space allowed for the internal representation of the `carray`. If params are correct and if the operation works, `ok` will hold the `carray` address; otherwise, it will hold `NULL`.

Parameters

<i>c</i>	the <code>carray</code>
<i>new_space</i>	new space (in nb of elements) for the <code>carray</code>
<i>validation</i>	flag

Definition at line 162 of file `carray.c`.

5.1.2.37 `type*` `carray_subarray_TF` (`carray * c`, `int from_index`, `int to_index`, `void ** ok`)

Returns a smaller vanilla array which holds values from index "`from_index`" (included) to index "`to_index`" (excluded). If params are correct and if the operation works, `ok` will hold the `carray` address; otherwise, it will hold `NULL`.

Parameters

<i>c</i>	the <code>carray</code>
<i>from_index</i>	beginning index
<i>to_index</i>	ending index
<i>ok</i>	validation flag

Returns

the sub-vanilla array if the indices are correct, NULL otherwise

Definition at line 977 of file carray.c.

5.1.2.38 `type* carray_subarraystep_TF (carray * c, int from_index, int to_index, int step, void ** ok)`

Returns a smaller vanilla array which holds values from index "from_index" (included) to index "to_index" (excluded) according to the specified step. If params are correct and if the operation works, ok will hold the carray address; otherwise, it will hold NULL.

Parameters

<i>c</i>	the carray
<i>from_index</i>	beginning index
<i>to_index</i>	ending index
<i>step</i>	item selecting step
<i>ok</i>	validation flag

Returns

the sub-vanilla array if the indices are correct, NULL otherwise

Definition at line 1019 of file carray.c.

5.1.2.39 `carray* carray_subcarray_TF (carray * c, int from_index, int to_index, void ** ok)`

Returns a smaller carray which holds values from index "from_index" (included) to index "to_index" (excluded). If params are correct and if the operation works, ok will hold the carray address; otherwise, it will hold NULL.

Parameters

<i>c</i>	the carray
<i>from_index</i>	beginning index
<i>to_index</i>	ending index
<i>ok</i>	validation flag

Returns

the sub-carray if the indices are correct, NULL otherwise

Definition at line 864 of file carray.c.

5.1.2.40 `carray* carray_subcarraystep_TF (carray * c, int from_index, int to_index, int step, void ** ok)`

Returns a smaller carray which holds values from index "from_index" (included) to index "to_index" (excluded) according to the specified step. If params are correct and if the operation works, ok will hold the carray address; otherwise, it will hold NULL.

Parameters

<i>c</i>	the carray
<i>from_index</i>	beginning index
<i>to_index</i>	ending index
<i>step</i>	item selecting step
<i>ok</i>	validation flag

Returns

the sub-carray if the indices are correct, NULL otherwise

Definition at line 908 of file carray.c.

5.1.2.41 `type* carray_toarray_TF (carray * c)`

Returns a vanilla array version of the carray.

Parameters

<i>c</i>	the carray
----------	------------

Returns

a vanilla array version of the carray

Definition at line 618 of file carray.c.

5.1.2.42 `char* carray_tostring_TF (carray * c, char (*)(type) strfunc, char * opener, char * closer, char * prefix, char * suffix)`

Returns a representation of the specified carray.

Parameters

<i>c</i>	the carray
<i>strfunc</i>	the function to apply on each element to convert it into string
<i>opener</i>	opening string
<i>closer</i>	closing string
<i>prefix</i>	prefix string added before each cell except for the first one
<i>suffix</i>	suffix string added after each cell except for the lats one

Returns

a string representing the carray

Definition at line 500 of file carray.c.

5.1.2.43 void fatal_error (char *message*[])

Prints an error message

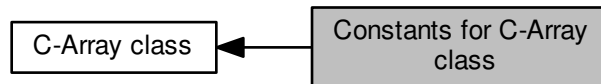
Parameters

<i>message</i>	message to print
----------------	------------------

Definition at line 27 of file carry.c.

5.2 Constants for C-Array class

Collaboration diagram for Constants for C-Array class:



Macros

- `#define DEFAULT_SPACE_INIT_FLAT 8`
- `#define DEFAULT_SPACE_INIT_PERCENT 1.25`
- `#define DEFAULT_SPACE_INCR 2.0`
- `#define DEFAULT_SHRINK_THRESHOLD 2.5`
- `#define DEFAULT_SHRINK_PERCENT 1.25`

Typedefs

- `typedef struct carray carray`

5.2.1 Detailed Description

5.2.2 Macro Definition Documentation

5.2.2.1 `#define DEFAULT_SHRINK_PERCENT 1.25`

Space factor used when the carray is shrunked.

Definition at line 50 of file carray.h.

5.2.2.2 `#define DEFAULT_SHRINK_THRESHOLD 2.5`

Space threshold from which the carray is shrunked.

Definition at line 45 of file carray.h.

5.2.2.3 `#define DEFAULT_SPACE_INCR 2.0`

Space factor used when a carray becomes too short.

Definition at line 40 of file carray.h.

5.2.2.4 `#define DEFAULT_SPACE_INIT_FLAT 8`

Initial space for the internal representation of a carray.

Definition at line 30 of file carray.h.

5.2.2.5 `#define DEFAULT_SPACE_INIT_PERCENT 1.25`

Initial space factor used when a carray is created from an existing one.

Definition at line 35 of file carray.h.

5.2.3 Typedef Documentation

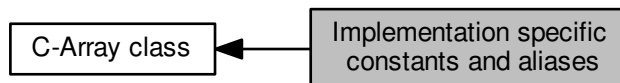
5.2.3.1 `typedef struct carray carray`

Alias for carray struct.

Definition at line 55 of file carray.h.

5.3 Implementation specific constants and aliases

Collaboration diagram for Implementation specific constants and aliases:



Macros

- `#define PRIME_CST 31`
- `#define DEFAULT_TYPE_VALUE NULL`

Typedefs

- `typedef unsigned int hashtype`
- `typedef void * type`

5.3.1 Detailed Description

5.4 Preprocessor macros

Collaboration diagram for Preprocessor macros:



Macros

- `#define of_Char *(char*)`
- `#define of_sChar *(signed char*)`
- `#define of_uChar *(unsigned char*)`
- `#define of_Short *(short*)`
- `#define of_sShort *(signed short*)`
- `#define of_uShort *(unsigned short*)`
- `#define of_Int *(int*)`
- `#define of_sInt *(signed int*)`
- `#define of_uInt *(unsigned int*)`
- `#define of_Long *(long*)`
- `#define of_ILong *(long long*)`
- `#define of_sLong *(signed long*)`
- `#define of_sILong *(signed long long*)`
- `#define of_uLong *(unsigned long*)`
- `#define of_uLong *(unsigned long long*)`
- `#define of_Float *(float*)`
- `#define of_Double *(double*)`
- `#define of_IDouble *(long double*)`
- `#define of_Bool *(bool*)`
- `#define of_String (char*)`
- `#define Char(expr)`
- `#define sChar(expr)`
- `#define uChar(expr)`
- `#define Short(expr)`
- `#define sShort(expr)`
- `#define uShort(expr)`
- `#define Int(expr)`
- `#define sInt(expr)`
- `#define uInt(expr)`
- `#define Long(expr)`
- `#define ILong(expr)`
- `#define sLong(expr)`
- `#define sILong(expr)`
- `#define uLong(expr)`
- `#define uLong(expr)`
- `#define Float(expr)`
- `#define Double(expr)`
- `#define IDouble(expr)`
- `#define Bool(expr)`
- `#define String(expr)`
- `#define free_Obj &ccarray_free_obj`

Functions

- void **carray_free_obj** (type)

Variables

- char * **Char_holder**
- signed char * **sChar_holder**
- unsigned char * **uChar_holder**
- short * **Short_holder**
- signed short * **sShort_holder**
- unsigned short * **uShort_holder**
- int * **Int_holder**
- signed int * **sInt_holder**
- unsigned int * **uInt_holder**
- long * **Long_holder**
- long long * **lLong_holder**
- signed long * **sLong_holder**
- signed long long * **slLong_holder**
- unsigned long * **uLong_holder**
- unsigned long long * **ulLong_holder**
- float * **Float_holder**
- double * **Double_holder**
- long double * **lDouble_holder**
- bool * **Bool_holder**
- char * **String_holder**

5.4.1 Detailed Description

5.4.2 Macro Definition Documentation

5.4.2.1 #define Bool(*expr*)

Value:

```
(Bool_holder = malloc(sizeof(bool)), \
 *Bool_holder = (expr), Bool_holder)
```

Definition at line 189 of file carray.h.

5.4.2.2 #define Char(*expr*)

Value:

```
(Char_holder = malloc(sizeof(char)), \
 *Char_holder = (expr), Char_holder)
```

Definition at line 99 of file carray.h.

5.4.2.3 #define Double(*expr*)

Value:

```
(Double_holder = malloc(sizeof(double)), \
 *Double_holder = (expr), Double_holder)
```

Definition at line 179 of file carray.h.

5.4.2.4 #define Float(*expr*)

Value:

```
(Float_holder = malloc(sizeof(float)), \
 *Float_holder = (expr), Float_holder)
```

Definition at line 174 of file carray.h.

5.4.2.5 #define Int(*expr*)

Value:

```
(Int_holder = malloc(sizeof(int)), \
 *Int_holder = (expr), Int_holder)
```

Definition at line 129 of file carray.h.

5.4.2.6 #define lDouble(*expr*)

Value:

```
(lDouble_holder = malloc(sizeof(long double)), \
 *lDouble_holder = (expr), lDouble_holder)
```

Definition at line 184 of file carray.h.

5.4.2.7 #define lLong(*expr*)

Value:

```
(lLong_holder = malloc(sizeof(long long)), \
 *lLong_holder = (expr), lLong_holder)
```

Definition at line 149 of file carray.h.

5.4.2.8 #define Long(*expr*)

Value:

```
(Long_holder = malloc(sizeof(long)), \
 *Long_holder = (expr), Long_holder)
```

Definition at line 144 of file carry.h.

5.4.2.9 #define sChar(*expr*)

Value:

```
(sChar_holder = malloc(sizeof(signed char)), \
 *sChar_holder = (expr), sChar_holder)
```

Definition at line 104 of file carry.h.

5.4.2.10 #define Short(*expr*)

Value:

```
(Short_holder = malloc(sizeof(short)), \
 *Short_holder = (expr), Short_holder)
```

Definition at line 114 of file carry.h.

5.4.2.11 #define sInt(*expr*)

Value:

```
(sInt_holder = malloc(sizeof(signed int)), \
 *sInt_holder = (expr), sInt_holder)
```

Definition at line 134 of file carry.h.

5.4.2.12 #define sLong(*expr*)

Value:

```
(sLong_holder = malloc(sizeof(signed long long)), \
 *sLong_holder = (expr), sLong_holder)
```

Definition at line 159 of file carry.h.

5.4.2.13 #define sLong(*expr*)

Value:

```
(sLong_holder = malloc(sizeof(signed long)), \
 *sLong_holder = (expr), sLong_holder)
```

Definition at line 154 of file carray.h.

5.4.2.14 #define sShort(*expr*)

Value:

```
(sShort_holder = malloc(sizeof(signed short)), \
 *sShort_holder = (expr), sShort_holder)
```

Definition at line 119 of file carray.h.

5.4.2.15 #define String(*expr*)

Value:

```
(String_holder = malloc(sizeof(expr) / sizeof(char)), \
 strcpy(String_holder, (expr)), \
 String_holder)
```

Definition at line 194 of file carray.h.

5.4.2.16 #define uChar(*expr*)

Value:

```
(uChar_holder = malloc(sizeof(unsigned char)), \
 *uChar_holder = (expr), uChar_holder)
```

Definition at line 109 of file carray.h.

5.4.2.17 #define ulnt(*expr*)

Value:

```
(uInt_holder = malloc(sizeof(unsigned int)), \
 *uInt_holder = (expr), uInt_holder)
```

Definition at line 139 of file carray.h.

5.4.2.18 #define uLong(*expr*)

Value:

```
(uLong_holder = malloc(sizeof(unsigned long long)), \
 *uLong_holder = (expr), uLong_holder)
```

Definition at line 169 of file `carray.h`.

5.4.2.19 #define uLong(*expr*)

Value:

```
(uLong_holder = malloc(sizeof(unsigned long)), \
 *uLong_holder = (expr), uLong_holder)
```

Definition at line 164 of file `carray.h`.

5.4.2.20 #define uShort(*expr*)

Value:

```
(uShort_holder = malloc(sizeof(unsigned short)), \
 *uShort_holder = (expr), uShort_holder)
```

Definition at line 124 of file `carray.h`.

5.4.3 Function Documentation

5.4.3.1 void `carray_free_obj` (type *val*)

Frees the specified primitive-wrapped element.

Parameters

<i>val</i>	primitive-wrapped element to free
------------	-----------------------------------

Definition at line 1077 of file `carray.c`.

5.5 C-Array core

Collaboration diagram for C-Array core:



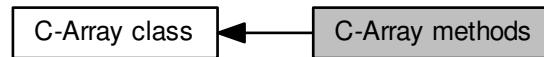
Data Structures

- struct **carray**

5.5.1 Detailed Description

5.6 C-Array methods

Collaboration diagram for C-Array methods:



Functions

- **carray * carray_new ()**
- **carray * carray_new_ISC (size_t)**
- **carray * carray_new_CC (carray *)**
- **carray * carray_new_CISC (carray *, size_t)**
- **void carray_free (carray *, void(type))**
- **size_t carray_getsize (carray *)**
- **void carray_clear (carray *)**
- **bool carray_isempty (carray *)**
- **size_t carray_getspace (carray *)**
- **void carray_setspace (carray *, size_t, void **)**
- **void carray_adjust (carray *, void **)**
- **size_t carray_getreadposition (carray *)**
- **size_t carray_readingsremaining (carray *)**
- **void carray_setreadposition (carray *, int, void **)**
- **type * carray_getarray (carray *)**
- **type carray_read (carray *, void **ok)**
- **void carray_reverse (carray *)**
- **carray * carray_reversed_TF (carray *)**
- **void carray_append (carray *, carray *, void **)**
- **carray * carray_concat_TF (carray *, carray *)**
- **char * carray_tostring_TF (carray *, char *(type), char *, char *, char *, char *)**
- **hashtype carray_hashcode (carray *, hashtype(type))**
- **bool carray_equal (carray *, carray *, bool(type, type))**
- **int carray_indexof (carray *, type, bool(type, type))**
- **int carray_lastindexof (carray *, type, bool(type, type))**
- **bool carray_contains (carray *, type, bool(type, type))**
- **carray * carray_subcarray_TF (carray *, int, int, void **)**
- **carray * carray_subcarraystep_TF (carray *, int, int, int, void **)**
- **type * carray_subarray_TF (carray *, int, int, void **)**
- **type * carray_subarraystep_TF (carray *, int, int, int, void **)**
- **type * carray_toarray_TF (carray *)**
- **type carray_get (carray *, int, void **)**
- **void carray_add (carray *, int, type, void **)**
- **void carray_push (carray *, type)**
- **void carray_ins (carray *, type)**
- **void carray_set (carray *, int, type, void **)**
- **void carray_safeset (carray *, int, type, void **)**
- **type carray_remove (carray *, int, void **)**
- **type carray_pop (carray *)**
- **bool carray_remove_elt (carray *, type, bool(type, type))**

5.6.1 Detailed Description

5.6.2 Function Documentation

5.6.2.1 void `carray_add` (`carray * c`, int `index`, type `new_value`, void ** `ok`)

Adds the specified element at the specified index in the carray. If params are correct and if the operation works, `ok` will hold the carray address; otherwise, it will hold NULL.

Parameters

<code>c</code>	the carray
<code>index</code>	index where to insert the specified element. The element will be inserted at this exact position, and subsequent elements will be shifted.
<code>new_value</code>	element to insert at the specified index
<code>ok</code>	validation flag

Definition at line 725 of file `carray.c`.

5.6.2.2 void `carray_adjust` (`carray * c`, void ** `ok`)

Eventually shrinks the size of the internal representation of the carray if free space exceed the threshold. `ok` will hold the carray address if a shrinking was made, and NULL otherwise.

Parameters

<code>c</code>	the carray
<code>ok</code>	validation flag

Definition at line 386 of file `carray.c`.

5.6.2.3 void `carray_append` (`carray * c`, `carray * o`, void ** `ok`)

Append the second specified carray at the end of the first one. If params are correct and if the operation works, `ok` will hold the carray address; otherwise, it will hold NULL.

Parameters

<code>c</code>	the carray
<code>o</code>	the carray to put at the end of the first one
<code>ok</code>	validation flag

Definition at line 251 of file `carray.c`.

5.6.2.4 void `carray_clear` (`carray * c`)

Clears the content of the carray. Be careful, this function DOESN'T free the space allowed for the internal representation of the carray. Use `carray_adjust` for that.

Parameters

<i>c</i>	the carray
----------	------------

Definition at line 652 of file `carray.c`.

5.6.2.5 `carray*` `carray_concat_TF` (`carray * a`, `carray * b`)

Concatenates both specified carrays into a new one which must be freed after use.

Parameters

<i>a</i>	the first carray to concatenate
<i>b</i>	the second carray to concatenate

Returns

a concatenated version of carrays *a* and *b*

Definition at line 438 of file `carray.c`.

5.6.2.6 `type` `carray_get` (`carray * c`, `int index`, `void ** ok`)

Gets the element at the specified index in the carray. If params are correct and if the operation works, *ok* will hold the carray address; otherwise, it will hold NULL.

Parameters

<i>c</i>	the carray
<i>index</i>	index where element will be retrieved
<i>ok</i>	validation flag

Returns

the element at the specified index if index is correct; default type value otherwise

Definition at line 667 of file `carray.c`.

5.6.2.7 `type*` `carray_getarray` (`carray * c`)

Internal array getter.

Parameters

<i>c</i>	the carray
----------	------------

Returns

the internal representation of the carray

Definition at line 320 of file carray.c.

5.6.2.8 `size_t carray_getreadposition (carray * c)`

Read position getter.

Parameters

<i>c</i>	the carray
----------	------------

Returns

the current read position

Definition at line 276 of file carray.c.

5.6.2.9 `size_t carray_getsize (carray * c)`

Size getter.

Parameters

<i>c</i>	the carray
----------	------------

Returns

the size of the carray

Definition at line 138 of file carray.c.

5.6.2.10 `size_t carray_getspace (carray * c)`

Space getter.

Parameters

<i>c</i>	the carray
----------	------------

Returns

the space (in nb of elements) of the carray

Definition at line 148 of file carray.c.

5.6.2.11 void carray_ins (carray * c, type value)

Inserts the specified value at the beginning of the carray.

Parameters

<i>c</i>	the carray
<i>value</i>	the element to insert at the end of the carray

Definition at line 362 of file carray.c.

5.6.2.12 bool carray_isempty (carray * c)

Returns true if the carray is empty, false otherwise

Parameters

<i>c</i>	the carray
----------	------------

Returns

true if the carray is empty, false otherwise

Definition at line 584 of file carray.c.

5.6.2.13 carray* carray_new ()

Constructor for carray. Returns a pointer to the created carray which must be freed after use.

Returns

a pointer to the created carray

Definition at line 37 of file carray.c.

5.6.2.14 carray* carray_new_CC (carray * copy_carray)

Copy constructor for carray. The created carray is the exact copy of the specified one with a bit more space than the length of the specified one. Returns a pointer to the created carray which must be freed after use.

Parameters

<i>copy_carray</i>	the carray to be copied
--------------------	-------------------------

Returns

a pointer to the created carray

Definition at line 76 of file carray.c.

5.6.2.15 carray* carray_new_CISC (carray * *copy_carray*, size_t *init_space*)

Copy constructor for carray with *init_space* specified. This carray have at least *init_space* slots at instantiation, and hold the exact content of the specified carray. Returns a pointer to the created carray which must be freed after use; if *init_space* is shorter than the length of the carray to be copied, returns NULL.

Parameters

<i>copy_carray</i>	the carray to be copied
<i>init_space</i>	Initial space (number of elements) of the carray

Returns

a pointer to the created carray

Definition at line 97 of file carray.c.

5.6.2.16 carray* carray_new_ISC (size_t *init_space*)

Constructor for carray with specified *init_space*. This carray have at least *init_space* slots at instantiation. Returns a pointer to the created carray which must be freed after use; if *init_space* is not valid, returns NULL.

Parameters

<i>init_space</i>	Initial space (number of elements) of the carray
-------------------	--

Returns

a pointer to the created carray

Definition at line 55 of file carray.c.

5.6.2.17 type carray_pop (carray * *c*)

Removes the last element of the carray and returns it.

Parameters

<i>c</i>	the carray
----------	------------

Returns

the last element of the carray

Definition at line 373 of file carray.c.

5.6.2.18 void carray_push (carray * *c*, type *value*)

Pushes the specified value at the end of the carray.

Parameters

<i>c</i>	the carray
<i>value</i>	the element to push at the end of the carray

Definition at line 351 of file carray.c.

5.6.2.19 type carray_read (carray * *c*, void ** *ok*)

Read method. Returns elt at the current read position and increases read position by 1. If params are correct and if the operation works, *ok* will hold the carray address; otherwise, it will hold NULL.

Parameters

<i>c</i>	the carray
<i>ok</i>	validation flag

Returns

the element at the current read position; default type value if there's no element to be read

Definition at line 335 of file carray.c.

5.6.2.20 size_t carray_readingsremaining (carray * *c*)

Gets the number of read operation remaining before reaching the end of the carray.

Parameters

<i>c</i>	the carray
----------	------------

Returns

the number of read possible before the end of the carray

Definition at line 310 of file carray.c.

5.6.2.21 type carray_remove (carray * c, int index, void ** ok)

Removes the element at the specified index and returns it. If params are correct and if the operation works, ok will hold the carray address; otherwise, it will hold NULL.

Parameters

<i>c</i>	the carray
<i>index</i>	index where element will be removed. The element will be removed at this exact position, and subsequent elements will be shifted.
<i>ok</i>	validation flag

Returns

the element which have been removed if the index is correct, default type value otherwise.

Definition at line 778 of file carray.c.

5.6.2.22 void carray_reverse (carray * c)

Reverses the specified carray in-place.

Parameters

<i>c</i>	the carray
----------	------------

Definition at line 402 of file carray.c.

5.6.2.23 carray* carray_reversed_TF (carray * c)

Returns a reversed copy of the specified carray which must be freed after use.

Parameters

<i>c</i>	the carray
----------	------------

Returns

a reversed copy of the specified carray

Definition at line 419 of file carray.c.

5.6.2.24 void `carray_safeset` (`carray * c`, `int index`, `type value`, `void ** ok`)

Used to set an element somewhere in the `carray` even if this cell is not already used by the `carray`, for example, to set `carray[8]` with a length-4 `carray`. Unused cells until the specified one are set with the default type value. If params are correct and if the operation works, `ok` will hold the `carray` address; otherwise, it will hold `NULL`.

Parameters

<i>c</i>	the <code>carray</code>
<i>index</i>	index where to put the specified value; can be larger than the <code>carray</code> size
<i>value</i>	value to be put at the specified index
<i>ok</i>	validation flag

Definition at line 207 of file `carray.c`.

5.6.2.25 void `carray_set` (`carray * c`, `int index`, `type new_value`, `void ** ok`)

Sets the element at the specified index in the `carray` to the specified value. If params are correct and if the operation works, `ok` will hold the `carray` address; otherwise, it will hold `NULL`.

Parameters

<i>c</i>	the <code>carray</code>
<i>index</i>	index where element will be set
<i>new_value</i>	new value for this element
<i>ok</i>	validation flag

Definition at line 697 of file `carray.c`.

5.6.2.26 void `carray_setreadposition` (`carray * c`, `int new_read_position`, `void ** ok`)

Read position setter. If params are correct and if the operation works, `ok` will hold the `carray` address; otherwise, it will hold `NULL`.

Parameters

<i>c</i>	the <code>carray</code>
<i>new_read_position</i>	new read position
<i>ok</i>	validation flag

Definition at line 289 of file `carray.c`.

5.6.2.27 void `carray_setspace` (`carray * c`, `size_t new_space`, `void ** ok`)

Space setter. Can be used to modify the space allowed for the internal representation of the `carray`. If params are correct and if the operation works, `ok` will hold the `carray` address; otherwise, it will hold `NULL`.

Parameters

<i>c</i>	the carray
<i>new_space</i>	new space (in nb of elements) for the carray
<i>validation</i>	flag

Definition at line 162 of file carray.c.

5.6.2.28 `type* carray_subarray_TF (carray * c, int from_index, int to_index, void ** ok)`

Returns a smaller vanilla array which holds values from index "from_index" (included) to index "to_index" (excluded). If params are correct and if the operation works, ok will hold the carray address; otherwise, it will hold NULL.

Parameters

<i>c</i>	the carray
<i>from_index</i>	beginning index
<i>to_index</i>	ending index
<i>ok</i>	validation flag

Returns

the sub-vanilla array if the indices are correct, NULL otherwise

Definition at line 977 of file carray.c.

5.6.2.29 `type* carray_subarraystep_TF (carray * c, int from_index, int to_index, int step, void ** ok)`

Returns a smaller vanilla array which holds values from index "from_index" (included) to index "to_index" (excluded) according to the specified step. If params are correct and if the operation works, ok will hold the carray address; otherwise, it will hold NULL.

Parameters

<i>c</i>	the carray
<i>from_index</i>	beginning index
<i>to_index</i>	ending index
<i>step</i>	item selecting step
<i>ok</i>	validation flag

Returns

the sub-vanilla array if the indices are correct, NULL otherwise

Definition at line 1019 of file carray.c.

5.6.2.30 `carray* carray_subcarray_TF (carray * c, int from_index, int to_index, void ** ok)`

Returns a smaller carray which holds values from index "from_index" (included) to index "to_index" (excluded). If params are correct and if the operation works, ok will hold the carray address; otherwise, it will hold NULL.

Parameters

<i>c</i>	the carray
<i>from_index</i>	beginning index
<i>to_index</i>	ending index
<i>ok</i>	validation flag

Returns

the sub-carrray if the indices are correct, NULL otherwise

Definition at line 864 of file carray.c.

5.6.2.31 `carray* carray_subcarraystep_TF (carray * c, int from_index, int to_index, int step, void ** ok)`

Returns a smaller carray which holds values from index "from_index" (included) to index "to_index" (excluded) according to the specified step. If params are correct and if the operation works, ok will hold the carray address; otherwise, it will hold NULL.

Parameters

<i>c</i>	the carray
<i>from_index</i>	beginning index
<i>to_index</i>	ending index
<i>step</i>	item selecting step
<i>ok</i>	validation flag

Returns

the sub-carrray if the indices are correct, NULL otherwise

Definition at line 908 of file carray.c.

5.6.2.32 `type* carray_toarray_TF (carray * c)`

Returns a vanilla array version of the carray.

Parameters

<i>c</i>	the carray
----------	------------

Returns

a vanilla array version of the carray

Definition at line 618 of file carray.c.

Chapter 6

Data Structure Documentation

6.1 carray Struct Reference

```
#include <carray.h>
```

Data Fields

- `type * _array`
- `size_t _size`
- `size_t _space`
- `size_t _read_position`

6.1.1 Detailed Description

Carray class implementation. Provides Python-like list for C.

Definition at line 216 of file `carray.h`.

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

- `carray.h`

Chapter 7

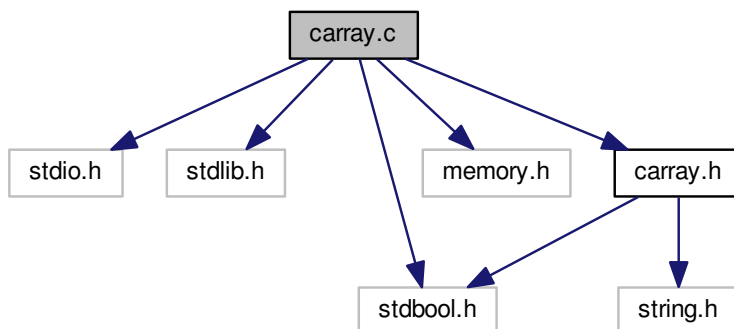
File Documentation

7.1 carray.c File Reference

Implementation of the carray class. Contains all the function implementations and documentation.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <memory.h>
#include "carray.h"
```

Include dependency graph for carray.c:



Functions

- void **fatal_error** (char message[])
- **carray** * **carray_new** ()
- **carray** * **carray_new_ISC** (size_t init_space)
- **carray** * **carray_new_CC** (**carray** *copy_carray)
- **carray** * **carray_new_CISC** (**carray** *copy_carray, size_t init_space)
- void **carray_free** (**carray** *c, void(*voidfunc)(type))

- `size_t carray_getsize (carray *c)`
- `size_t carray_getspace (carray *c)`
- `void carray_setspace (carray *c, size_t new_space, void **ok)`
- `void carray_addspace (carray *c, void **ok)`
- `void carray_safeset (carray *c, int index, type value, void **ok)`
- `void carray_append (carray *c, carray *o, void **ok)`
- `size_t carray_getreadposition (carray *c)`
- `void carray_setreadposition (carray *c, int new_read_position, void **ok)`
- `size_t carray_readingsremaining (carray *c)`
- `type * carray_getarray (carray *c)`
- `type carray_read (carray *c, void **ok)`
- `void carray_push (carray *c, type value)`
- `void carray_ins (carray *c, type value)`
- `type carray_pop (carray *c)`
- `void carray_adjust (carray *c, void **ok)`
- `void carray_reverse (carray *c)`
- `carray * carray_reversed_TF (carray *c)`
- `carray * carray_concat_TF (carray *a, carray *b)`
- `hashtype carray_hashcode (carray *c, hashtype(*hashfunc)(type))`
- `bool carray_equal (carray *a, carray *b, bool(*eqfunc)(type, type))`
- `char * carray_tostring_TF (carray *c, char *(*strfunc)(type), char *opener, char *closer, char *prefix, char *suffix)`
- `bool carray_isempty (carray *c)`
- `bool carray_contains (carray *c, type test_element, bool(*eqfunc)(type, type))`
- `type * carray_toarray_TF (carray *c)`
- `bool carray_remove_elt (carray *c, type test_element, bool(*eqfunc)(type, type))`
- `void carray_clear (carray *c)`
- `type carray_get (carray *c, int index, void **ok)`
- `void carray_set (carray *c, int index, type new_value, void **ok)`
- `void carray_add (carray *c, int index, type new_value, void **ok)`
- `type carray_remove (carray *c, int index, void **ok)`
- `int carray_indexof (carray *c, type test_value, bool(*eqfunc)(type, type))`
- `int carray_lastindexof (carray *c, type test_value, bool(*eqfunc)(type, type))`
- `carray * carray_subcarray_TF (carray *c, int from_index, int to_index, void **ok)`
- `carray * carray_subcarraystep_TF (carray *c, int from_index, int to_index, int step, void **ok)`
- `type * carray_subarray_TF (carray *c, int from_index, int to_index, void **ok)`
- `type * carray_subarraystep_TF (carray *c, int from_index, int to_index, int step, void **ok)`
- `void carray_free_obj (type val)`

7.1.1 Detailed Description

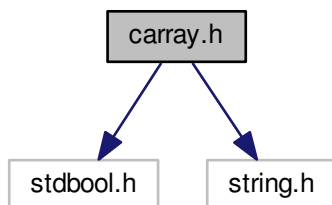
Implementation of the carray class. Contains all the function implementations and documentation.

7.2 carray.h File Reference

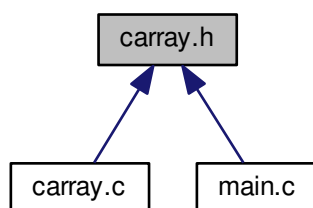
Header of the carray class. Contains all function declarations and preprocessor directives.

```
#include <stdbool.h>
#include <string.h>
```

Include dependency graph for carray.h:



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



Data Structures

- struct **carray**

Macros

- #define **DEFAULT_SPACE_INIT_FLAT** 8
- #define **DEFAULT_SPACE_INIT_PERCENT** 1.25
- #define **DEFAULT_SPACE_INCR** 2.0
- #define **DEFAULT_SHRINK_THRESHOLD** 2.5
- #define **DEFAULT_SHRINK_PERCENT** 1.25
- #define **PRIME_CST** 31
- #define **DEFAULT_TYPE_VALUE** NULL
- #define **of_Char** *(char*)
- #define **of_sChar** *(signed char*)
- #define **of_uChar** *(unsigned char*)
- #define **of_Short** *(short*)
- #define **of_sShort** *(signed short*)

- `#define of_uShort *(unsigned short*)`
- `#define of_Int *(int*)`
- `#define of_sInt *(signed int*)`
- `#define of_uInt *(unsigned int*)`
- `#define of_Long *(long*)`
- `#define of_ILong *(long long*)`
- `#define of_sLong *(signed long*)`
- `#define of_sILong *(signed long long*)`
- `#define of_uLong *(unsigned long*)`
- `#define of_uILong *(unsigned long long*)`
- `#define of_Float *(float*)`
- `#define of_Double *(double*)`
- `#define of_IDouble *(long double*)`
- `#define of_Bool *(bool*)`
- `#define of_String (char*)`
- `#define Char(expr)`
- `#define sChar(expr)`
- `#define uChar(expr)`
- `#define Short(expr)`
- `#define sShort(expr)`
- `#define uShort(expr)`
- `#define Int(expr)`
- `#define sInt(expr)`
- `#define uInt(expr)`
- `#define Long(expr)`
- `#define ILong(expr)`
- `#define sLong(expr)`
- `#define sILong(expr)`
- `#define uLong(expr)`
- `#define uILong(expr)`
- `#define Float(expr)`
- `#define Double(expr)`
- `#define IDouble(expr)`
- `#define Bool(expr)`
- `#define String(expr)`
- `#define free_Obj &carray_free_obj`

Typedefs

- `typedef struct carray carray`
- `typedef unsigned int hashtype`
- `typedef void * type`

Functions

- `void carray_free_obj (type)`
- `carray * carray_new ()`
- `carray * carray_new_ISC (size_t)`
- `carray * carray_new_CC (carray *)`
- `carray * carray_new_CISC (carray *, size_t)`
- `void carray_free (carray *, void(type))`
- `size_t carray_getsize (carray *)`
- `void carray_clear (carray *)`

- `bool carray_isempty (carray *)`
- `size_t carray_getspace (carray *)`
- `void carray_setspace (carray *, size_t, void **)`
- `void carray_adjust (carray *, void **)`
- `size_t carray_getreadposition (carray *)`
- `size_t carray_readingsremaining (carray *)`
- `void carray_setreadposition (carray *, int, void **)`
- `type * carray_getarray (carray *)`
- `type carray_read (carray *, void **ok)`
- `void carray_reverse (carray *)`
- `carray * carray_reversed_TF (carray *)`
- `void carray_append (carray *, carray *, void **)`
- `carray * carray_concat_TF (carray *, carray *)`
- `char * carray_tostring_TF (carray *, char *(type), char *, char *, char *, char *)`
- `hashtype carray_hashcode (carray *, hashtype(type))`
- `bool carray_equal (carray *, carray *, bool(type, type))`
- `int carray_indexof (carray *, type, bool(type, type))`
- `int carray_lastindexof (carray *, type, bool(type, type))`
- `bool carray_contains (carray *, type, bool(type, type))`
- `carray * carray_subcarray_TF (carray *, int, int, void **)`
- `carray * carray_subcarraystep_TF (carray *, int, int, int, void **)`
- `type * carray_subarray_TF (carray *, int, int, void **)`
- `type * carray_subarraystep_TF (carray *, int, int, int, void **)`
- `type * carray_toarray_TF (carray *)`
- `type carray_get (carray *, int, void **)`
- `void carray_add (carray *, int, type, void **)`
- `void carray_push (carray *, type)`
- `void carray_ins (carray *, type)`
- `void carray_set (carray *, int, type, void **)`
- `void carray_safeset (carray *, int, type, void **)`
- `type carray_remove (carray *, int, void **)`
- `type carray_pop (carray *)`
- `bool carray_remove_elt (carray *, type, bool(type, type))`

Variables

- `char * Char_holder`
- `signed char * sChar_holder`
- `unsigned char * uChar_holder`
- `short * Short_holder`
- `signed short * sShort_holder`
- `unsigned short * uShort_holder`
- `int * Int_holder`
- `signed int * sInt_holder`
- `unsigned int * uInt_holder`
- `long * Long_holder`
- `long long * lLong_holder`
- `signed long * sLong_holder`
- `signed long long * slLong_holder`
- `unsigned long * uLong_holder`
- `unsigned long long * ulLong_holder`
- `float * Float_holder`
- `double * Double_holder`
- `long double * lDouble_holder`
- `bool * Bool_holder`
- `char * String_holder`

7.2.1 Detailed Description

Header of the carry class. Contains all function declarations and preprocessor directives.

Index

Bool

Preprocessor macros, 29

C-Array class, 9

- carray_add, 10
- carray_addspace, 11
- carray_adjust, 11
- carray_append, 11
- carray_clear, 11
- carray_concat_TF, 12
- carray_contains, 12
- carray_equal, 12
- carray_free, 13
- carray_free_obj, 13
- carray_get, 13
- carray_getarray, 14
- carray_getreadposition, 14
- carray_getsize, 14
- carray_getspace, 14
- carray_hashcode, 15
- carray_indexof, 15
- carray_ins, 15
- carray_isempty, 16
- carray_lastindexof, 16
- carray_new, 16
- carray_new_CISC, 17
- carray_new_CC, 17
- carray_new_ISC, 17
- carray_pop, 18
- carray_push, 18
- carray_read, 18
- carray_readingsremaining, 18
- carray_remove, 19
- carray_remove_elt, 19
- carray_reverse, 19
- carray_reversed_TF, 20
- carray_safeset, 20
- carray_set, 20
- carray_setreadposition, 21
- carray_setspace, 21
- carray_subarray_TF, 21
- carray_subarraystep_TF, 22
- carray_subcarray_TF, 22
- carray_subcarraystep_TF, 22
- carray_toarray_TF, 23
- carray_tostring_TF, 23
- fatal_error, 23

C-Array core, 34

C-Array methods, 35

carray_add, 36

carray_adjust, 36

carray_append, 36

carray_clear, 36

carray_concat_TF, 37

carray_get, 37

carray_getarray, 37

carray_getreadposition, 38

carray_getsize, 38

carray_getspace, 38

carray_ins, 39

carray_isempty, 39

carray_new, 39

carray_new_CISC, 40

carray_new_CC, 39

carray_new_ISC, 40

carray_pop, 40

carray_push, 41

carray_read, 41

carray_readingsremaining, 41

carray_remove, 42

carray_reverse, 42

carray_reversed_TF, 42

carray_safeset, 42

carray_set, 43

carray_setreadposition, 43

carray_setspace, 43

carray_subarray_TF, 44

carray_subarraystep_TF, 44

carray_subcarray_TF, 44

carray_subcarraystep_TF, 45

carray_toarray_TF, 45

carray, 47

Constants for C-Array class, 26

carray.c, 49

carray.h, 50

carray_add

C-Array class, 10

C-Array methods, 36

carray_addspace

C-Array class, 11

carray_adjust

C-Array class, 11

C-Array methods, 36

carray_append

C-Array class, 11

C-Array methods, 36

carray_clear

C-Array class, 11

C-Array methods, 36

- carray_concat_TF
 - C-Array class, 12
 - C-Array methods, 37
- carray_contains
 - C-Array class, 12
- carray_equal
 - C-Array class, 12
- carray_free
 - C-Array class, 13
- carray_free_obj
 - C-Array class, 13
 - Preprocessor macros, 33
- carray_get
 - C-Array class, 13
 - C-Array methods, 37
- carray_getarray
 - C-Array class, 14
 - C-Array methods, 37
- carray_getreadposition
 - C-Array class, 14
 - C-Array methods, 38
- carray_getsize
 - C-Array class, 14
 - C-Array methods, 38
- carray_getspace
 - C-Array class, 14
 - C-Array methods, 38
- carray_hashcode
 - C-Array class, 15
- carray_indexof
 - C-Array class, 15
- carray_ins
 - C-Array class, 15
 - C-Array methods, 39
- carray_isempty
 - C-Array class, 16
 - C-Array methods, 39
- carray_lastindexof
 - C-Array class, 16
- carray_new
 - C-Array class, 16
 - C-Array methods, 39
- carray_new_CISC
 - C-Array class, 17
 - C-Array methods, 40
- carray_new_CC
 - C-Array class, 17
 - C-Array methods, 39
- carray_new_ISC
 - C-Array class, 17
 - C-Array methods, 40
- carray_pop
 - C-Array class, 18
 - C-Array methods, 40
- carray_push
 - C-Array class, 18
 - C-Array methods, 41
- carray_read
 - C-Array class, 18
 - C-Array methods, 41
- carray_readingsremaining
 - C-Array class, 18
 - C-Array methods, 41
- carray_remove
 - C-Array class, 19
 - C-Array methods, 42
- carray_remove_elt
 - C-Array class, 19
- carray_reverse
 - C-Array class, 19
 - C-Array methods, 42
- carray_reversed_TF
 - C-Array class, 20
 - C-Array methods, 42
- carray_safeset
 - C-Array class, 20
 - C-Array methods, 42
- carray_set
 - C-Array class, 20
 - C-Array methods, 43
- carray_setreadposition
 - C-Array class, 21
 - C-Array methods, 43
- carray_setspace
 - C-Array class, 21
 - C-Array methods, 43
- carray_subarray_TF
 - C-Array class, 21
 - C-Array methods, 44
- carray_subarraystep_TF
 - C-Array class, 22
 - C-Array methods, 44
- carray_subcarray_TF
 - C-Array class, 22
 - C-Array methods, 44
- carray_subcarraystep_TF
 - C-Array class, 22
 - C-Array methods, 45
- carray_toarray_TF
 - C-Array class, 23
 - C-Array methods, 45
- carray_tostring_TF
 - C-Array class, 23
- Char
 - Preprocessor macros, 29
- Constants for C-Array class, 25
 - carray, 26
 - DEFAULT_SHRINK_PERCENT, 25
 - DEFAULT_SHRINK_THRESHOLD, 25
 - DEFAULT_SPACE_INCR, 25
 - DEFAULT_SPACE_INIT_FLAT, 25
 - DEFAULT_SPACE_INIT_PERCENT, 26
- DEFAULT_SHRINK_PERCENT
 - Constants for C-Array class, 25
- DEFAULT_SHRINK_THRESHOLD
 - Constants for C-Array class, 25

- DEFAULT_SPACE_INCR
 - Constants for C-Array class, 25
- DEFAULT_SPACE_INIT_FLAT
 - Constants for C-Array class, 25
- DEFAULT_SPACE_INIT_PERCENT
 - Constants for C-Array class, 26
- Double
 - Preprocessor macros, 29
- fatal_error
 - C-Array class, 23
- Float
 - Preprocessor macros, 30
- Implementation specific constants and aliases, 27
- Int
 - Preprocessor macros, 30
- IDouble
 - Preprocessor macros, 30
- ILong
 - Preprocessor macros, 30
- Long
 - Preprocessor macros, 30
- Preprocessor macros, 28
 - Bool, 29
 - carray_free_obj, 33
 - Char, 29
 - Double, 29
 - Float, 30
 - Int, 30
 - IDouble, 30
 - ILong, 30
 - Long, 30
 - sChar, 31
 - sInt, 31
 - sLong, 31
 - sShort, 32
 - Short, 31
 - slLong, 31
 - String, 32
 - uChar, 32
 - uInt, 32
 - uLong, 33
 - uShort, 33
 - ulLong, 32
- sChar
 - Preprocessor macros, 31
- sInt
 - Preprocessor macros, 31
- sLong
 - Preprocessor macros, 31
- sShort
 - Preprocessor macros, 32
- Short
 - Preprocessor macros, 31
- slLong
 - Preprocessor macros, 31
- String
 - Preprocessor macros, 32
- uChar
 - Preprocessor macros, 32
- uInt
 - Preprocessor macros, 32
- uLong
 - Preprocessor macros, 33
- uShort
 - Preprocessor macros, 33
- ulLong
 - Preprocessor macros, 32