

Almog Dynamic Array

Generated by Doxygen 1.9.1

1 README	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Class Documentation	7
4.1 ada_float_array Struct Reference	7
4.1.1 Detailed Description	7
4.1.2 Member Data Documentation	7
4.1.2.1 capacity	7
4.1.2.2 elements	7
4.1.2.3 length	8
4.2 ada_int_array Struct Reference	8
4.2.1 Detailed Description	8
4.2.2 Member Data Documentation	8
4.2.2.1 capacity	8
4.2.2.2 elements	8
4.2.2.3 length	8
5 File Documentation	9
5.1 Almog_Dynamic_Array.h File Reference	9
5.1.1 Detailed Description	10
5.1.2 Macro Definition Documentation	11
5.1.2.1 ada_appand	11
5.1.2.2 ADA_ASSERT	12
5.1.2.3 ADA_EXIT	12
5.1.2.4 ada_init_array	12
5.1.2.5 ADA_INIT_CAPACITY	13
5.1.2.6 ada_insert	13
5.1.2.7 ada_insert_unordered	14
5.1.2.8 ADA_MALLOC	15
5.1.2.9 ADA_REALLOC	15
5.1.2.10 ada_remove	15
5.1.2.11 ada_remove_unordered	16
5.1.2.12 ada_resize	17
5.2 Almog_Dynamic_Array.h	18
5.3 README.md File Reference	19
5.4 test.c File Reference	19
5.4.1 Macro Definition Documentation	20
5.4.1.1 ADA_FLOAT_PRINT	20
5.4.1.2 ADA_INT_PRINT	20

5.4.2 Function Documentation	20
5.4.2.1 main()	20
5.4.2.2 print_float_ada()	21
5.4.2.3 print_int_ada()	21
5.5 test.c	21
Index	23

Chapter 1

README

Works with structs. For example:

```
typedef struct {
    size_t length;
    size_t capacity;
    int* elements;
} ada_int_array;
```


Chapter 2

Class Index

2.1 Class List

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

ada_float_array	7
ada_int_array	8

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Almog_Dynamic_Array.h	Header-only C macros that implement a simple dynamic array	9
test.c	19

Chapter 4

Class Documentation

4.1 ada_float_array Struct Reference

Public Attributes

- size_t `length`
- size_t `capacity`
- float * `elements`

4.1.1 Detailed Description

Definition at line 10 of file [test.c](#).

4.1.2 Member Data Documentation

4.1.2.1 capacity

`size_t ada_float_array::capacity`

Definition at line 12 of file [test.c](#).

Referenced by [print_float_ada\(\)](#).

4.1.2.2 elements

`float* ada_float_array::elements`

Definition at line 13 of file [test.c](#).

Referenced by [print_float_ada\(\)](#).

4.1.2.3 length

size_t ada_float_array::length

Definition at line 11 of file [test.c](#).

Referenced by [print_float_ada\(\)](#).

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

- [test.c](#)

4.2 ada_int_array Struct Reference

Public Attributes

- size_t [length](#)
- size_t [capacity](#)
- int * [elements](#)

4.2.1 Detailed Description

Definition at line 4 of file [test.c](#).

4.2.2 Member Data Documentation

4.2.2.1 capacity

size_t ada_int_array::capacity

Definition at line 6 of file [test.c](#).

Referenced by [print_int_ada\(\)](#).

4.2.2.2 elements

int* ada_int_array::elements

Definition at line 7 of file [test.c](#).

Referenced by [print_int_ada\(\)](#).

4.2.2.3 length

size_t ada_int_array::length

Definition at line 5 of file [test.c](#).

Referenced by [print_int_ada\(\)](#).

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

- [test.c](#)

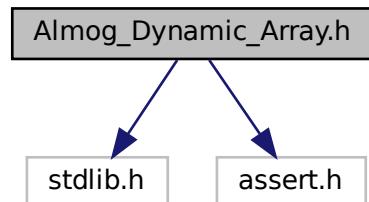
Chapter 5

File Documentation

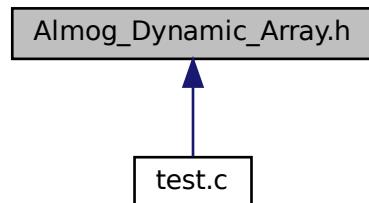
5.1 Almog_Dynamic_Array.h File Reference

Header-only C macros that implement a simple dynamic array.

```
#include <stdlib.h>
#include <assert.h>
Include dependency graph for Almog_Dynamic_Array.h:
```



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



Macros

- `#define ADA_INIT_CAPACITY 10`
Default initial capacity used by ada_init_array.
- `#define ADA_MALLOC malloc`
Allocation function used by this header (defaults to malloc).
- `#define ADA_EXIT exit`
- `#define ADA_REALLOC realloc`
Reallocation function used by this header (defaults to realloc).
- `#define ADA_ASSERT assert`
Assertion macro used by this header (defaults to assert).
- `#define ada_init_array(type, header)`
Initialize an array header and allocate its initial storage.
- `#define ada_resize(type, header, new_capacity)`
Resize the underlying storage to hold new_capacity elements.
- `#define ada_appand(type, header, value)`
Append a value to the end of the array, growing if necessary.
- `#define ada_insert(type, header, value, index)`
Insert value at position index, preserving order ($O(n)$).
- `#define ada_insert_unordered(type, header, value, index)`
Insert value at index without preserving order ($O(1)$ amortized).
- `#define ada_remove(type, header, index)`
Remove element at index, preserving order ($O(n)$).
- `#define ada_remove_unordered(type, header, index)`
Remove element at index by moving the last element into its place ($O(1)$); order is not preserved.

5.1.1 Detailed Description

Header-only C macros that implement a simple dynamic array.

This header provides a minimal, macro-based dynamic array for POD-like types. The array "header" is a user-defined struct with three fields:

- `size_t length; current number of elements`
- `size_t capacity; allocated capacity (in elements)`
- `T* elements; pointer to contiguous storage of elements (type T)`

How to use: 1) Define a header struct with length/capacity/elements fields. 2) Initialize it with `ada_init_array(T, header)`. 3) Modify it with `ada_appand` (append), `ada_insert`, `remove` variants, etc. 4) When done, free(header.elements) (or your custom deallocator).

Customization:

- Define `ADA_MALLOC`, `ADA_REALLOC`, and `ADA_ASSERT` before including this header to override allocation and assertion behavior.

Complexity (n = number of elements):

- Append: amortized $O(1)$

- Ordered insert/remove: O(n)
- Unordered insert/remove: O(1)

Notes and limitations:

- These are macros; arguments may be evaluated multiple times. Pass only simple lvalues (no side effects).
- Index checks rely on ADA_ASSERT; with NDEBUG they may be compiled out.
- ada_resize exits the process (exit(1)) if reallocation fails.
- ada_insert reads header.elements[header.length - 1] internally; inserting into an empty array via ada_insert is undefined behavior. Use ada_append or ada_insert_unordered for that case.
- No automatic shrinking; you may call ada_resize manually.

Example: `typedef struct { size_t length; size_t capacity; int* elements; } ada_int_array;`

```
ada_int_array arr; ada_init_array(int, arr); ada_append(int, arr, 42); ada_insert(int, arr, 7, 0); // requires arr.length > 0
ada_remove(int, arr, 1); free(arr.elements);
```

Definition in file [Almog_Dynamic_Array.h](#).

5.1.2 Macro Definition Documentation

5.1.2.1 ada_append

```
#define ada_append(
    type,
    header,
    value )
```

Value:

```
do {
    if ((header).length >= (header).capacity) {
        ada_resize(type, (header), (int)((header).capacity + (header).capacity/2 + 1));
    }
    (header).elements[(header).length] = value;
    (header).length++;
} while (0)
```

Append a value to the end of the array, growing if necessary.

Parameters

<code>type</code>	Element type stored in the array.
<code>header</code>	Lvalue of the header struct.
<code>value</code>	Value to append.

Postcondition

header.length is incremented by 1; the last element equals value.

Note

Growth factor is (int)(header.capacity * 1.5). Because of truncation, very small capacities may not grow (e.g., from 1 to 1). With the default INIT_CAPACITY=10 this is typically not an issue unless you manually shrink capacity. Ensure growth always increases capacity by at least 1 if you customize this macro.

Definition at line 176 of file [Almog_Dynamic_Array.h](#).

5.1.2.2 ADA_ASSERT

```
#define ADA_ASSERT assert
```

Assertion macro used by this header (defaults to assert).

Define ADA_ASSERT before including this file to override. When NDEBUG is defined, standard assert() is disabled.

Definition at line 103 of file [Almog_Dynamic_Array.h](#).

5.1.2.3 ADA_EXIT

```
#define ADA_EXIT exit
```

Definition at line 79 of file [Almog_Dynamic_Array.h](#).

5.1.2.4 ada_init_array

```
#define ada_init_array( \
    type, \
    header )
```

Value:

```
do { \
    (header).capacity = ADA_INIT_CAPACITY; \
    (header).length = 0; \
    (header).elements = (type *)ADA_MALLOC(sizeof(type) * (header).capacity); \
    ADA_ASSERT((header).elements != NULL); \
} while (0)
```

Initialize an array header and allocate its initial storage.

Parameters

<i>type</i>	Element type stored in the array (e.g., int).
<i>header</i>	Lvalue of the header struct containing fields: length, capacity, and elements.

Precondition

header is a modifiable lvalue; header.elements is uninitialized or ignored and will be overwritten.

Postcondition

header.length == 0, header.capacity == INIT_CAPACITY, header.elements != NULL (or ADA_ASSERT fails).

Note

Allocation uses ADA_MALLOC and is checked via ADA_ASSERT.

Definition at line 127 of file [Almog_Dynamic_Array.h](#).

5.1.2.5 ADA_INIT_CAPACITY

```
#define ADA_INIT_CAPACITY 10
```

Default initial capacity used by ada_init_array.

You may override this by defining ADA_INIT_CAPACITY before including this file.

Definition at line 62 of file [Almog_Dynamic_Array.h](#).

5.1.2.6 ada_insert

```
#define ada_insert(
    type,
    header,
    value,
    index )
```

Value:

```
do {
    ADA_ASSERT((int)(index) >= 0);
    ADA_ASSERT((float)(index) - (int)(index) == 0);
    ada_append(type, (header), (header).elements[(header).length-1]);
    for (int ada_for_loop_index = (header).length-2; ada_for_loop_index > (int)(index);
        ada_for_loop_index--) {
        (header).elements[ada_for_loop_index] = (header).elements [ada_for_loop_index-1];
    }
    (header).elements[(index)] = value;
} while (0)
```

Insert value at position index, preserving order ($O(n)$).

Parameters

<i>type</i>	Element type stored in the array.
<i>header</i>	Lvalue of the header struct.
<i>value</i>	Value to insert.
<i>index</i>	Destination index in the range [0, header.length].

Precondition

$0 \leq \text{index} \leq \text{header.length}$.

$\text{header.length} > 0$ if $\text{index} == \text{header.length}$ (this macro reads the last element internally). For inserting into an empty array, use `ada_append` or `ada_insert_unordered`.

Postcondition

Element is inserted at index; subsequent elements are shifted right; `header.length` is incremented by 1.

Note

This macro asserts index is non-negative and an integer value using `ADA_ASSERT`. No explicit upper-bound assert is performed.

Definition at line 203 of file [Almog_Dynamic_Array.h](#).

5.1.2.7 ada_insert_unordered

```
#define ada_insert_unordered(                                     \
    type,                                                 \
    header,                                              \
    value,                                               \
    index )
```

Value:

```
do { \
    ADA_ASSERT((int)(index) >= 0); \
    ADA_ASSERT((float)(index) - (int)(index) == 0); \
    if ((size_t)(index) == (header).length) { \
        ada_append(type, (header), value); \
    } else { \
        ada_append(type, (header), (header).elements[(index)]); \
        (header).elements[(index)] = value; \
    } \
} while (0)
```

Insert value at index without preserving order ($O(1)$ amortized).

If $\text{index} == \text{header.length}$, this behaves like an append. Otherwise, the current element at index is moved to the end, and value is written at index.

Parameters

<i>type</i>	Element type stored in the array.
<i>header</i>	Lvalue of the header struct.
<i>value</i>	Value to insert.
<i>index</i>	Index in the range [0, header.length].

Precondition

$0 \leq index \leq header.length$.

Postcondition

$header.length$ is incremented by 1; array order is not preserved.

Definition at line 229 of file [Almog_Dynamic_Array.h](#).

5.1.2.8 ADA_MALLOC

```
#define ADA_MALLOC malloc
```

Allocation function used by this header (defaults to malloc).

Define ADA_MALLOC to a compatible allocator before including this file to override the default.

Definition at line 74 of file [Almog_Dynamic_Array.h](#).

5.1.2.9 ADA_REALLOC

```
#define ADA_REALLOC realloc
```

Reallocation function used by this header (defaults to realloc).

Define ADA_REALLOC to a compatible reallocator before including this file to override the default.

Definition at line 91 of file [Almog_Dynamic_Array.h](#).

5.1.2.10 ada_remove

```
#define ada_remove(
    type,
    header,
    index )
```

Value:

```
do {
    ADA_ASSERT((int)(index) >= 0);
    ADA_ASSERT((float)(index) - (int)(index) == 0);
    for (size_t ada_for_loop_index = (index); ada_for_loop_index < (header).length-1; ada_for_loop_index++)
    {
        (header).elements[ada_for_loop_index] = (header).elements[ada_for_loop_index+1];
    }
    (header).length--;
} while (0)
```

Remove element at index, preserving order ($O(n)$).

Parameters

<i>type</i>	Element type stored in the array.
<i>header</i>	Lvalue of the header struct.
<i>index</i>	Index in the range [0, header.length - 1].

Precondition

$0 \leq index < header.length$.

Postcondition

header.length is decremented by 1; subsequent elements are shifted left by one position. The element beyond the new length is left uninitialized.

Definition at line 253 of file [Almog_Dynamic_Array.h](#).

5.1.2.11 ada_remove_unordered

```
#define ada_remove_unordered(
    type,
    header,
    index )
```

Value:

```
do {           \
    ADA_ASSERT((int)(index) >= 0);          \
    ADA_ASSERT((float)(index) - (int)(index) == 0); \
    (header).elements[index] = (header).elements[(header).length-1]; \
    (header).length--;                         \
} while (0)
```

Remove element at index by moving the last element into its place ($O(1)$); order is not preserved.

Parameters

<i>type</i>	Element type stored in the array.
<i>header</i>	Lvalue of the header struct.
<i>index</i>	Index in the range [0, header.length - 1].

Precondition

$0 \leq index < header.length$ and $header.length > 0$.

Postcondition

header.length is decremented by 1; array order is not preserved.

Definition at line 274 of file [Almog_Dynamic_Array.h](#).

5.1.2.12 ada_resize

```
#define ada_resize(
    type,
    header,
    new_capacity )
```

Value:

```
do {
    type *ada_temp_pointer = (type *)ADA_REALLOC((void *)((header).elements),
new_capacity*sizeof(type)); \
    if (ada_temp_pointer == NULL) { \
        ADA_EXIT(1); \
    } \
    (header).elements = ada_temp_pointer; \
    ADA_ASSERT((header).elements != NULL); \
    (header).capacity = new_capacity;
} while (0)
```

Resize the underlying storage to hold new_capacity elements.

Parameters

<i>type</i>	Element type stored in the array.
<i>header</i>	Lvalue of the header struct.
<i>new_capacity</i>	New capacity in number of elements.

Precondition

new_capacity >= *header.length* (otherwise elements beyond *new_capacity* are lost and *length* will not be adjusted).

Postcondition

header.capacity == *new_capacity* and *header.elements* points to a block large enough for *new_capacity* elements.

Warning

On allocation failure, this macro calls [ADA_EXIT\(1\)](#).

Note

Reallocation uses ADA_REALLOC and is also checked via ADA_ASSERT.

Definition at line 150 of file [Almog_Dynamic_Array.h](#).

5.2 Almog_Dynamic_Array.h

```

00001
00051 #ifndef ALMOG_DYNAMIC_ARRAY_H_
00052 #define ALMOG_DYNAMIC_ARRAY_H_
00053
00054
00061 #ifndef ADA_INIT_CAPACITY
00062 #define ADA_INIT_CAPACITY 10
00063 #endif /*ADA_INIT_CAPACITY*/
00064
00072 #ifndef ADA_MALLOC
00073 #include <stdlib.h>
00074 #define ADA_MALLOC malloc
00075 #endif /*ADA_MALLOC*/
00076
00077 #ifndef ADA_EXIT
00078 #include <stdlib.h>
00079 #define ADA_EXIT exit
00080 #endif /*ADA_EXIT*/
00081
00089 #ifndef ADA_REALLOC
00090 #include <stdlib.h>
00091 #define ADA_REALLOC realloc
00092 #endif /*ADA_REALLOC*/
00093
00101 #ifndef ADA_ASSERT
00102 #include <assert.h>
00103 #define ADA_ASSERT assert
00104 #endif /*ADA_ASSERT*/
00105
00106 /* typedef struct {
00107     size_t length;
00108     size_t capacity;
00109     int* elements;
00110 } ada_int_array; */
00111
00127 #define ada_init_array(type, header) do {
00128     (header).capacity = ADA_INIT_CAPACITY;
00129     (header).length = 0;
00130     (header).elements = (type *)ADA_MALLOC(sizeof(type) * (header).capacity);
00131     ADA_ASSERT((header).elements != NULL);
00132 } while (0)
00133
00150 #define ada_resize(type, header, new_capacity) do {
00151     type *ada_temp_pointer = (type *)ADA_REALLOC((void *)((header).elements),
00152     new_capacity*sizeof(type));
00153     if (ada_temp_pointer == NULL) {
00154         ADA_EXIT(1);
00155     }
00156     (header).elements = ada_temp_pointer;
00157     ADA_ASSERT((header).elements != NULL);
00158     (header).capacity = new_capacity;
00159 } while (0)
00159
00176 #define ada_appand(type, header, value) do {
00177     if ((header).length >= (header).capacity) {
00178         ada_resize(type, (header), (int)((header).capacity + (header).capacity/2 + 1));
00179     }
00180     (header).elements[(header).length] = value;
00181     (header).length++;
00182 } while (0)
00183
00203 #define ada_insert(type, header, value, index) do {
00204     ADA_ASSERT((int)(index) >= 0);
00205     ADA_ASSERT((float)(index) - (int)(index) == 0);
00206     ada_appand(type, (header), (header).elements[(header).length-1]);
00207     for (int ada_for_loop_index = (header).length-2; ada_for_loop_index > (int)(index);
00208         ada_for_loop_index--) {
00209             (header).elements[ada_for_loop_index] = (header).elements [ada_for_loop_index-1];
00210         }
00211     }
00211 } while (0)

```

```

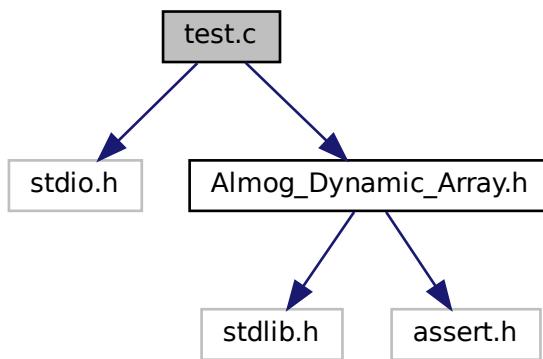
00212
00213
00229 #define ada_insert_unordered(type, header, value, index) do { \
00230     ADA_ASSERT((int)(index) >= 0); \
00231     ADA_ASSERT((float)(index) - (int)(index) == 0); \
00232     if ((size_t)(index) == (header).length) { \
00233         ada_append(type, (header), value); \
00234     } else { \
00235         ada_append(type, (header), (header).elements[(index)]); \
00236         (header).elements[(index)] = value; \
00237     } \
00238 } while (0)
00239
00253 #define ada_remove(type, header, index) do { \
00254     ADA_ASSERT((int)(index) >= 0); \
00255     ADA_ASSERT((float)(index) - (int)(index) == 0); \
00256     for (size_t ada_for_loop_index = (index); ada_for_loop_index < (header).length-1; \
00257         ada_for_loop_index++) { \
00258         (header).elements[ada_for_loop_index] = (header).elements[ada_for_loop_index+1]; \
00259     } \
00260     (header).length--; \
00261 } while (0)
00262
00274 #define ada_remove_unordered(type, header, index) do { \
00275     ADA_ASSERT((int)(index) >= 0); \
00276     ADA_ASSERT((float)(index) - (int)(index) == 0); \
00277     (header).elements[index] = (header).elements[(header).length-1]; \
00278     (header).length--; \
00279 } while (0)
00280
00281
00282 #endif /*ALMOG_DYNAMIC_ARRAY_H*/

```

5.3 README.md File Reference

5.4 test.c File Reference

```
#include <stdio.h>
#include "Almog_Dynamic_Array.h"
Include dependency graph for test.c:
```



Classes

- struct `ada_int_array`
- struct `ada_float_array`

Macros

- `#define ADA_INT_PRINT(ada) print_int_ada(ada, #ada)`
- `#define ADA_FLOAT_PRINT(ada) print_float_ada(ada, #ada)`

Functions

- void `print_int_ada (ada_int_array ada, char *name)`
- void `print_float_ada (ada_float_array ada, char *name)`
- int `main ()`

5.4.1 Macro Definition Documentation

5.4.1.1 ADA_FLOAT_PRINT

```
#define ADA_FLOAT_PRINT(  
    ada ) print_float_ada (ada, #ada)
```

Definition at line [46](#) of file `test.c`.

5.4.1.2 ADA_INT_PRINT

```
#define ADA_INT_PRINT(  
    ada ) print_int_ada (ada, #ada)
```

Definition at line [30](#) of file `test.c`.

5.4.2 Function Documentation

5.4.2.1 main()

```
int main ( )
```

Definition at line [48](#) of file `test.c`.

References `ada_appand`, `ADA_FLOAT_PRINT`, `ada_init_array`, `ada_insert`, and `ADA_INT_PRINT`.

5.4.2.2 print_float_ada()

```
void print_float_ada (
    ada_float_array ada,
    char * name )
```

Definition at line 32 of file [test.c](#).

References [ada_float_array::capacity](#), [ada_float_array::elements](#), and [ada_float_array::length](#).

5.4.2.3 print_int_ada()

```
void print_int_ada (
    ada_int_array ada,
    char * name )
```

Definition at line 16 of file [test.c](#).

References [ada_int_array::capacity](#), [ada_int_array::elements](#), and [ada_int_array::length](#).

5.5 test.c

```
00001 #include <stdio.h>
00002 #include "Almog_Dynamic_Array.h"
00003
00004 typedef struct {
00005     size_t length;
00006     size_t capacity;
00007     int* elements;
00008 } ada_int_array;
00009
00010 typedef struct {
00011     size_t length;
00012     size_t capacity;
00013     float* elements;
00014 } ada_float_array;
00015
00016 void print_int_ada(ada_int_array ada, char *name)
00017 {
00018     printf("%s\n", name);
00019     printf("capacity: %zu\n", ada.capacity);
00020     printf("length: %zu\n", ada.length);
00021     if (ada.length == 0) {
00022         printf("]\n\n");
00023         return;
00024     }
00025     for (size_t i = 0; i < ada.length - 1; i++) {
00026         printf("%d, ", ada.elements[i]);
00027     }
00028     printf("%d]\n\n", ada.elements[ada.length - 1]);
00029 }
00030 #define ADA_INT_PRINT(ada) print_int_ada(ada, #ada)
00031
00032 void print_float_ada(ada_float_array ada, char *name)
00033 {
00034     printf("%s\n", name);
00035     printf("capacity: %zu\n", ada.capacity);
00036     printf("length: %zu\n", ada.length);
00037     if (ada.length == 0) {
00038         printf("]\n\n");
00039         return;
00040     }
00041     for (size_t i = 0; i < ada.length - 1; i++) {
00042         printf("%g, ", ada.elements[i]);
00043     }
00044     printf("%g]\n\n", ada.elements[ada.length - 1]);
00045 }
00046 #define ADA_FLOAT_PRINT(ada) print_float_ada(ada, #ada)
```

```
00047
00048 int main()
00049 {
00050     ada_int_array a;
00051
00052     ada_init_array(int, a);
00053
00054     for (int i = 0; i < 14; i++) {
00055         ada_appand(int, a, i);
00056     }
00057
00058     ADA_INT_PRINT(a);
00059
00060     ada_insert(int, a, 100, 1);
00061     ada_insert(int, a, 100, 1);
00062     ADA_INT_PRINT(a);
00063
00064
00065     ada_float_array b;
00066
00067     ada_init_array(float, b);
00068
00069     for (int i = 0; i < 69; i++) {
00070         ada_appand(float, b, i/2.0);
00071     }
00072
00073     ADA_FLOAT_PRINT(b);
00074
00075
00076     return 0;
00077 }
00078
```

Index

ada_appand
 Almog_Dynamic_Array.h, 11

ADA_ASSERT
 Almog_Dynamic_Array.h, 12

ADA_EXIT
 Almog_Dynamic_Array.h, 12

ada_float_array, 7
 capacity, 7
 elements, 7
 length, 7

ADA_FLOAT_PRINT
 test.c, 20

ada_init_array
 Almog_Dynamic_Array.h, 12

ADA_INIT_CAPACITY
 Almog_Dynamic_Array.h, 13

ada_insert
 Almog_Dynamic_Array.h, 13

ada_insert_unordered
 Almog_Dynamic_Array.h, 14

ada_int_array, 8
 capacity, 8
 elements, 8
 length, 8

ADA_INT_PRINT
 test.c, 20

ADA_MALLOC
 Almog_Dynamic_Array.h, 15

ADA_REALLOC
 Almog_Dynamic_Array.h, 15

ada_remove
 Almog_Dynamic_Array.h, 15

ada_remove_unordered
 Almog_Dynamic_Array.h, 16

ada_resize
 Almog_Dynamic_Array.h, 16

Almog_Dynamic_Array.h, 9
 ada_appand, 11
 ADA_ASSERT, 12
 ADA_EXIT, 12
 ada_init_array, 12
 ADA_INIT_CAPACITY, 13
 ada_insert, 13
 ada_insert_unordered, 14
 ADA_MALLOC, 15
 ADA_REALLOC, 15
 ada_remove, 15
 ada_remove_unordered, 16
 ada_resize, 16

capacity
 ada_float_array, 7
 ada_int_array, 8

elements
 ada_float_array, 7
 ada_int_array, 8

length
 ada_float_array, 7
 ada_int_array, 8

main
 test.c, 20

print_float_ada
 test.c, 20

print_int_ada
 test.c, 21

README.md, 19

test.c, 19
 ADA_FLOAT_PRINT, 20
 ADA_INT_PRINT, 20
 main, 20
 print_float_ada, 20
 print_int_ada, 21