

# Almog Dynamic Array

Generated by Doxygen 1.9.1



---

<b>1 README</b>	<b>1</b>
<b>2 Class Index</b>	<b>3</b>
2.1 Class List . . . . .	3
<b>3 File Index</b>	<b>5</b>
3.1 File List . . . . .	5
<b>4 Class Documentation</b>	<b>7</b>
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
<b>5 File Documentation</b>	<b>9</b>
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_init_array . . . . .	12
5.1.2.4 ADA_INIT_CAPACITY . . . . .	13
5.1.2.5 ada_insert . . . . .	13
5.1.2.6 ada_insert_unordered . . . . .	14
5.1.2.7 ADA_MALLOC . . . . .	15
5.1.2.8 ADA_REALLOC . . . . .	15
5.1.2.9 ada_remove . . . . .	15
5.1.2.10 ada_remove_unordered . . . . .	16
5.1.2.11 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
<b>Index</b>	<b>23</b>

# 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:

<a href="#">Almog_Dynamic_Array.h</a>	Header-only C macros that implement a simple dynamic array . . . . .	9
<a href="#">test.c</a>	.....	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](#)

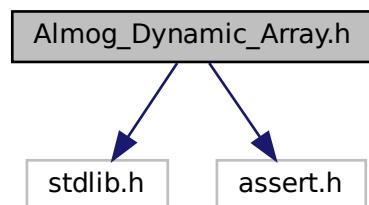
# Chapter 5

## File Documentation

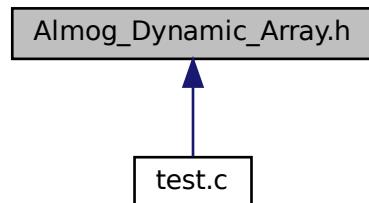
### 5.1 Almog\_Dynamic\_Array.h File Reference

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_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_append(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_append` (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*1.5));
    }
    header.elements[header.length] = value;
    header.length++;
} while (0)
```

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

#### Parameters

<i>type</i>	Element type stored in the array.
<i>header</i>	Lvalue of the header struct.
<i>value</i>	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 170 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 97 of file [Almog\\_Dynamic\\_Array.h](#).

**5.1.2.3 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 121 of file [Almog\\_Dynamic\\_Array.h](#).

### 5.1.2.4 ADA\_INIT\_CAPACITY

```
#define ADA_INIT_CAPACITY 10
```

Default initial capacity used by `ada_init_array`.

You may override this by defining INIT\_CAPACITY before including this file.

Definition at line 62 of file [Almog\\_Dynamic\\_Array.h](#).

### 5.1.2.5 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 (size_t ada_for_loop_index = header.length-2; ada_for_loop_index > (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

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

**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 197 of file [Almog\\_Dynamic\\_Array.h](#).

**5.1.2.6 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 $[0, \text{header.length}]$ .

**Precondition**

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

### Postcondition

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

Definition at line 223 of file [Almog\\_Dynamic\\_Array.h](#).

### 5.1.2.7 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 73 of file [Almog\\_Dynamic\\_Array.h](#).

### 5.1.2.8 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 85 of file [Almog\\_Dynamic\\_Array.h](#).

### 5.1.2.9 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 247 of file [Almog\\_Dynamic\\_Array.h](#).

**5.1.2.10 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 268 of file [Almog\\_Dynamic\\_Array.h](#).

### 5.1.2.11 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) {
        \
        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*  $\geq$  *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 `exit(1)`.

**Note**

Reallocation uses `ADA_REALLOC` and is also checked via `ADA_ASSERT`.

Definition at line 144 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
00055
00062 #define ADA_INIT_CAPACITY 10
00063
00071 #ifndef ADA_MALLOC
00072 #include <stdlib.h>
00073 #define ADA_MALLOC malloc
00074 #endif /*ADA_MALLOC*/
00075
00083 #ifndef ADA_REALLOC
00084 #include <stdlib.h>
00085 #define ADA_REALLOC realloc
00086 #endif /*ADA_REALLOC*/
00087
00095 #ifndef ADA_ASSERT
00096 #include <assert.h>
00097 #define ADA_ASSERT assert
00098 #endif /*ADA_ASSERT*/
00099
00100 /* typedef struct {
00101     size_t length;
00102     size_t capacity;
00103     int* elements;
00104 } ada_int_array; */
00105
00121 #define ada_init_array(type, header) do {
00122     header.capacity = ADA_INIT_CAPACITY;
00123     header.length = 0;
00124     header.elements = (type *)ADA_MALLOC(sizeof(type) * header.capacity);
00125     ADA_ASSERT(header.elements != NULL);
00126 } while (0)
00127
00144 #define ada_resize(type, header, new_capacity) do {
00145     type *ada_temp_pointer = (type *)ADA_REALLOC((void *)(header.elements),
00146 new_capacity*sizeof(type)); \
00146     if (ada_temp_pointer == NULL) {
00147         \
00148         exit(1);
00149     \
00150     header.elements = ada_temp_pointer;
00151     \
00152     ADA_ASSERT(header.elements != NULL);
00153     \
00154     header.capacity = new_capacity;
00155 } while (0)
00156
00170 #define ada_appand(type, header, value) do {
00171     if (header.length >= header.capacity) {
00172         ada_resize(type, header, (int)(header.capacity*1.5));
00173     }
00174     header.elements[header.length] = value;
00175     header.length++;
00176 } while (0)
00177
00197 #define ada_insert(type, header, value, index) do {
00198     ADA_ASSERT((int)(index) >= 0);
00199     ADA_ASSERT((float)(index) - (int)(index) == 0);
00200     ada_appand(type, header, header.elements[header.length-1]);
00201     for (size_t ada_for_loop_index = header.length-2; ada_for_loop_index > (index);
00202 ada_for_loop_index--) {
00203         header.elements[ada_for_loop_index] = header.elements [ada_for_loop_index-1];
00204     }
00205     header.elements[(index)] = value;
00206 } while (0)
00207
00223 #define ada_insert_unordered(type, header, value, index) do {
00224     ADA_ASSERT((int)(index) >= 0);
00225     ADA_ASSERT((float)(index) - (int)(index) == 0);
00226     if ((size_t)(index) == header.length) {
```

```

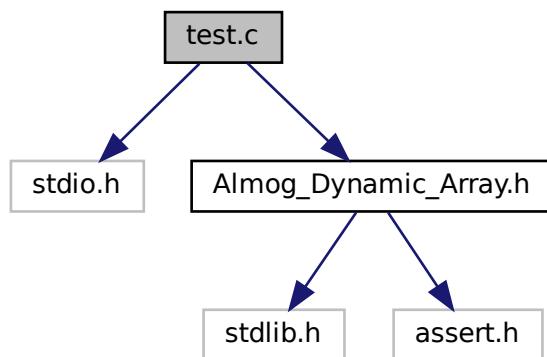
00227     ada_append(type, header, value);
00228 } else {
00229     ada_append(type, header, header.elements[(index)]);
00230     header.elements[(index)] = value;
00231 }
00232 } while (0)
00233
00247 #define ada_remove(type, header, index) do {
00248     \
00249     ADA_ASSERT((int)(index) >= 0);
00250     \
00251     ADA_ASSERT((float)(index) - (int)(index) == 0);
00252     \
00253     for (size_t ada_for_loop_index = (index); ada_for_loop_index < header.length-1;
00254         ada_for_loop_index++) {
00255         \
00256         header.elements[ada_for_loop_index] = header.elements[ada_for_loop_index+1];
00257         \
00258     }
00259     \
00260     header.length--;
00261 }
00262 } while (0)
00263
00264 #define ada_remove_unordered(type, header, index) do {
00265     \
00266     ADA_ASSERT((int)(index) >= 0);
00267     \
00268     ADA_ASSERT((float)(index) - (int)(index) == 0);
00269     \
00270     header.elements[index] = header.elements[header.length-1];
00271     \
00272     header.length--;
00273 } while (0)
00274
00275
00276 #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\_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\_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