Linear data structures using C

0.2

Generated by Doxygen 1.8.17

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Module Documentation

4.1 Dynamic array

Implementation dynamic array (vector).

Functions

- void vector_foreach (vector_t *v, void(*f)(void *))

  Applies a function to each element of the array.
- vector_t vector_filter (vector_t *v, int(*f)(void *))

  Add to a new array elements whose predicate returns true.
- int vector_any (vector_t *v, int(*f)(void *))

  Return true if at least one element satisfies the predicate.
- void vector_fold (vector_t *v, void *acc, void(*fun)(void *, void *))

  Executes the function once for each array element.
- void vector_bubble_sort (vector_t *v, int(*cmp)(void *, void *))

  Implementation bubble sort algorithm.
- void vector_insertion_sort (vector_t *v, int(*cmp)(void *, void *))

  Implementation insertion sort algorithm.
- void vector_init (vector_t *v, size_t capacity, size_t elem_size)

  Construct new vector data structure.
- void vector_null (vector_t *v)

  Implementation null object pattern.
- void vector_free (vector_t *v, void(*deleter)(void *))

  Free memory dynamic array.
- void vector_push_back (vector_t *v, void *elem)

  Adds an item to the end.
- void vector_insert_by_index (vector_t *v, size_t index, void *elem)

  Adds an item by index. If index > size then don't add.
- void vector_delete_by_value (vector_t *v, void *key, int(*cmp)(void *, void *), void(*deleter)(void *))

  Delete item by value, shifting all trailing elements left.
- void vector_delete_by_index (vector_t *v, size_t index, void(*deleter)(void *))

  Delete item at index, shifting all trailing elements left.
- void * vector_get (vector_t *v, size_t index)

  Returns item at given index.

- void vector_set (vector_t *v, size_t index, void *elem)

  Change element by index.
- size_t vector_size (vector_t *v)

  Returns size of vector (expressed in terms of elements).
- size_t vector_is_empty (vector_t *v)

  Returns true if vector is empty (expressed in terms of elements).
- long long vector_find (vector_t *v, void *key, int(*cmp)(void *, void *))

  Implementation linear search.
- long long vector_binary_search (vector_t *v, void *key, int(*cmp)(void *, void *))

  Implementation binary search.

4.1.1 Detailed Description

Implementation dynamic array (vector).

Authors

Dosart

Version

0.2.0

Date

25 april 2021

Warning

This structure created only for educational goals.

4.1.2 Function Documentation

4.1.2.1 vector_any()

```
int vector_any (
            vector_t * v,
            int(*)(void *) f )
```

Return true if at least one element satisfies the predicate.

Parameters

v	pointer to vector.
f	function predicate. Applies to every item v.

Returns

bool.

4.1.2.2 vector_binary_search()

```
long long vector_binary_search (
            vector_t * v,
            void * key,
            int(*)(void *, void *) cmp )
```

Implementation binary search.

Parameters

v	Pointer to vector data structure
key	Key for search
cmp	A function that takes two arguments (key and current item of vector). Returns EQUAL if the key are equal, returns LESS if key less, returns MORE if key more.

Returns

index of find element Returns -1 if key is not found.

4.1.2.3 vector_bubble_sort()

```
void vector_bubble_sort (
            vector_t * v,
            int(*)(void *, void *) cmp )
```

Implementation bubble sort algorithm.

Warning

Doesn't check for NULL equality.

Parameters

v	pointer to vector.
cmp	comparison function (returns True if x y and False otherwise).

4.1.2.4 vector_delete_by_index()

```
void vector_delete_by_index (
            vector_t * v,
            size_t index,
            void(*)(void *) deleter )
```

Delete item at index, shifting all trailing elements left.

Parameters

v	Pointer to vector data structure.
index	Index for delete.
deleter	Function to remove an item.

4.1.2.5 vector_delete_by_value()

```
void vector_delete_by_value (
            vector_t * v,
            void * key,
            int(*)(void *, void *) cmp,
            void(*)(void *) deleter )
```

Delete item by value, shifting all trailing elements left.

Parameters

v	Pointer to vector data structure.
key	Value for delete.
cmp	The function that takes two arguments. Returns 1 if the elements are.
deleter	The function to remove an item.

4.1.2.6 vector_filter()

```
vector_t vector_filter (
            vector_t * v,
            int(*)(void *) f )
```

Add to a new array elements whose predicate returns true.

Parameters

v	pointer to vector.
f	function predicate. Applies to every item v.

Returns

New vector with filtered values.

4.1.2.7 vector_find()

```
long long vector_find (
             vector_t * v,
             void * key,
             int(*)(void *, void *) cmp )
```

Implementation linear search.

Parameters

v	Pointer to vector data structure.
key	Key for search
cmp	A function that takes two arguments (key and current item of vector). Returns 1 if the elements are equal, otherwise 0.

Returns

index of find element Returns -1 if key is not found.

4.1.2.8 vector_fold()

```
void vector_fold (
             vector_t * v,
             void * acc,
             void(*)(void *, void *) fun )
```

Executes the function once for each array element.

Parameters

v	pointer to vector.
f	function. Applies to every item v.

4.1.2.9 vector_foreach()

```
void vector_foreach (
             vector_t * v,
             void(*)(void *) f )
```

Applies a function to each element of the array.

Parameters

v	pointer to vector.
f	function.

4.1.2.10 vector_free()

```
void vector_free (
            vector_t * v,
            void(*)(void *) deleter )
```

Free memory dynamic array.

Parameters

v	Pointer to vector.
deleter	Function to remove an item.

4.1.2.11 vector_get()

```
void* vector_get (
            vector_t * v,
            size_t index )
```

Returns item at given index.

Parameters

v	Pointer to vector data structure.
elem	Element for add.

Returns

item At given index.

4.1.2.12 vector_init()

```
void vector_init (
            vector_t * v,
            size_t capacity,
            size_t elem_size )
```

Construct new vector data structure.

Parameters

v	Pointer to vector data structure.
capacity	The size of the storage space currently allocated for the vector, expressed in terms of elements. if capacity == 0, set capacity = 16 on first addition.
elem_size	Size of vector item.

4.1.2.13 vector_insert_by_index()

```
void vector_insert_by_index (
            vector_t * v,
            size_t index,
            void * elem )
```

Adds an item by index. If index > size then don't add.

Parameters

v	Pointer to vector data structure.
index	Index for added.
elem	Element for add.

4.1.2.14 vector_insertion_sort()

```
void vector_insertion_sort (
            vector_t * v,
            int(*)(void *, void *) cmp )
```

Implementation insertion sort algorithm.

Warning

Doesn't check for NULL equality.

Parameters

v	pointer to vector.
cmp	comparison function (returns True if x y and False otherwise).

4.1.2.15 vector_is_empty()

```
size_t vector_is_empty (
```

```
            vector_t * v )
```

Returns true if vector is empty (expressed in terms of elements).

Parameters

v	Pointer to vector data structure.

Returns

size_t 1 if size == 0 else 0.

4.1.2.16 vector_push_back()

```
void vector_push_back (
            vector_t * v,
            void * elem )
```

Adds an item to the end.

Parameters

v	Pointer to vector data structure.
elem	Element for add.

4.1.2.17 vector_set()

```
void vector_set (
            vector_t * v,
            size_t index,
            void * elem )
```

Change element by index.

Parameters

v	Pointer to vector data structure.
index	Index for change element.
elem	Element for change.

Returns

item At given index.

4.1.2.18 vector_size()

```
size_t vector_size (
            vector_t * v )
```

Returns size of vector (expressed in terms of elements).

Parameters

v	Pointer to vector data structure.

Returns

size Size of vector.

4.2 Stack

Item of stack.

Classes

- struct stack_t

  Implementation stack using linked list.

Typedefs

- typedef struct stack_t stack_t

  Implementation stack using linked list.

Functions

- void stack_init (stack_t *v, size_t elem_size)

  Construct new stack data structure.
- size_t stack_is_empty (stack_t *s)

  Returns true if stack is empty (expressed in terms of elements).
- size_t stack_size (stack_t *s)

  Returns size of stack (expressed in terms of elements).
- void stack_free (stack_t *s, void(*deleter)(void *))

  Free memory in stack_item_t.
- void stack_push (stack_t *s, void *item)

  Add item to stack.
- void * stack_peek (stack_t *s)

  Looks at the object at the top of this s without removing it from the s.
- void stack_pop (stack_t *s, void(*deleter)(void *))

  Removes the object at the top of this stack and returns that object as the value of this function.

4.2.1 Detailed Description

Item of stack.

Authors

Dosart

Version

0.2.0

Date

07 May 2021

Warning

This structure created only for educational goals

4.2.2 Typedef Documentation

4.2.2.1 stack_t

```
typedef struct stack_t stack_t
```

Implementation stack using linked list.

Authors

Dosart

Version

0.2.0

Date

07 May 2021

Warning

This structure created only for educational goals

4.2.3 Function Documentation

4.2.3.1 stack_free()

```
void stack_free (
            stack_t * s,
            void(*)(void *) deleter )
```

Free memory in stack_item_t.

Parameters

s	Pointer to stack.
deleter	Deleter function to remove an item.

4.2.3.2 stack_init()

```
void stack_init (
            stack_t * v,
            size_t elem_size )
```

Construct new stack data structure.

Parameters

s	Pointer to stack data structure.
elem_size	Size of stack item.

4.2.3.3 stack_is_empty()

```
size_t stack_is_empty (
            stack_t * s )
```

Returns true if stack is empty (expressed in terms of elements).

Parameters

s	Pointer to stack data structure.

Returns

1 if size == 0 else 0.

4.2.3.4 stack_peek()

```
void* stack_peek (
            stack_t * s )
```

Looks at the object at the top of this s without removing it from the s.

Parameters

s	Pointer to s.

Returns

Top of this s.

4.2.3.5 stack_pop()

```
void stack_pop (
            stack_t * s,
            void(*)(void *) deleter )
```

Removes the object at the top of this stack and returns that object as the value of this function.

Parameters

s	Pointer to s.

Returns

Top of this s.

Parameters

deleter	Deleter function to remove an item.

4.2.3.6 stack_push()

```
void stack_push (
            stack_t * s,
            void * item )
```

Add item to stack.

Parameters

s	pointer to stack.
item	element for add.

4.2.3.7 stack_size()

```
size_t stack_size (
            stack_t * s )
```

Returns size of stack (expressed in terms of elements).

Parameters

s	Pointer to stack data structure.

Returns

Size of stack.

Chapter 5

Class Documentation

5.1 point Struct Reference

Public Attributes

- int **x**
- int **y**

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

- linear_data_structures/stack/tests/test.c

5.2 search_test_item Struct Reference

Public Attributes

- long long(∗ **search**)(vector_t ∗, void ∗, int(∗cmp)(void ∗, void ∗))
- int(∗ **cmp**)(void ∗, void ∗)
- char ∗ **message**

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

- linear_data_structures/vector/tests/test_search.c

5.3 sort_test_item Struct Reference

Public Attributes

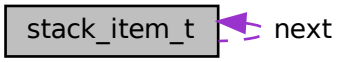
- void(∗ **sort**)(vector_t ∗v, int(∗cmp)(void ∗, void ∗))
- int(∗ **cmp**)(void ∗, void ∗)
- char ∗ **message**

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

- linear_data_structures/vector/tests/test_sort.c

5.4 stack_item_t Struct Reference

Collaboration diagram for stack_item_t:



Public Attributes

- void * **data**
- struct stack_item_t * **next**

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

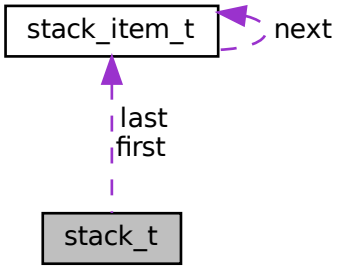
- linear_data_structures/stack/include/stack.h

5.5 stack_t Struct Reference

Implementation stack using linked list.

`#include <stack.h>`

Collaboration diagram for stack_t:



Public Attributes

- stack_item_t * **first**
- stack_item_t * **last**
- size_t elem_size

  size of one element
- size_t size

  count of elements on the stack

5.5.1 Detailed Description

Implementation stack using linked list.

Authors

Dosart

Version

0.2.0

Date

07 May 2021

Warning

This structure created only for educational goals

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

- linear_data_structures/stack/include/stack.h

5.6 vector_t Struct Reference

Public Attributes

- void * **data**
- size_t **size**
- size_t **capacity**
- size_t **elem_size**

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

- linear_data_structures/vector/include/vector.h

Chapter 6

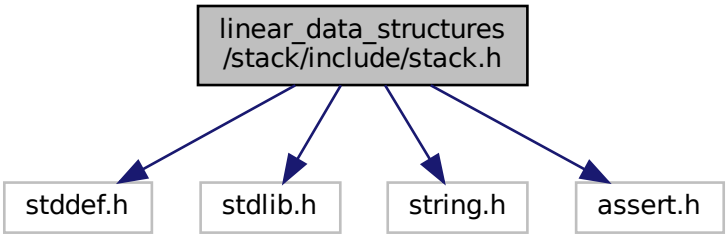
File Documentation

6.1 linear_data_structures/stack/include/stack.h File Reference

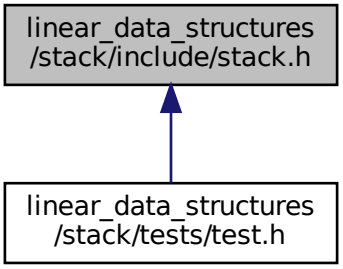
Header file for stack.

```
#include <stddef.h>
#include <stdlib.h>
#include <string.h>
#include "assert.h"
```

Include dependency graph for stack.h:



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



Classes

- struct stack_item_t
- struct stack_t

  Implementation stack using linked list.

Macros

- #define **STACK_FOREACH**(stack, function)

Typedefs

- typedef struct stack_item_t **stack_item_t**
- typedef struct stack_t stack_t

  Implementation stack using linked list.

Functions

- void stack_init (stack_t *v, size_t elem_size)

  Construct new stack data structure.
- size_t stack_is_empty (stack_t *s)

  Returns true if stack is empty (expressed in terms of elements).
- size_t stack_size (stack_t *s)

  Returns size of stack (expressed in terms of elements).
- void stack_free (stack_t *s, void(*deleter)(void *))

  Free memory in stack_item_t.
- void stack_push (stack_t *s, void *item)

  Add item to stack.
- void * stack_peek (stack_t *s)

  Looks at the object at the top of this s without removing it from the s.
- void stack_pop (stack_t *s, void(*deleter)(void *))

  Removes the object at the top of this stack and returns that object as the value of this function.

6.1.1 Detailed Description

Header file for stack.

This file contains the definition of the data structure stack

6.1.2 Macro Definition Documentation

6.1.2.1 STACK_FOREACH

```
#define STACK_FOREACH(
              stack,
              function )
```

Value:

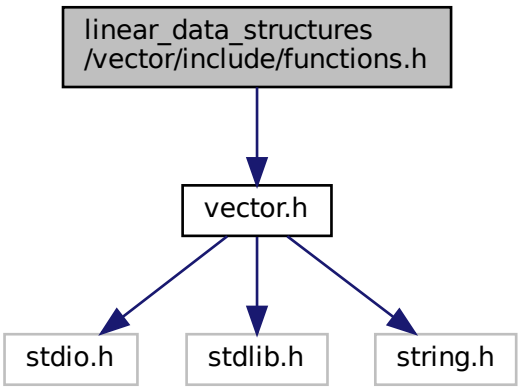
```
  { \
  stack_item_t* curr = stack->first;      \
  while(curr) {                           \
    function(curr->data);                 \
    curr = curr->next;                   \
  }\
}
```

6.2 linear_data_structures/vector/include/functions.h File Reference

Header file for implement higher order functions.

```
#include "vector.h"
```

Include dependency graph for functions.h:



Functions

- void vector_foreach (vector_t *v, void(*f)(void *))

  Applies a function to each element of the array.
- vector_t vector_filter (vector_t *v, int(*f)(void *))

  Add to a new array elements whose predicate returns true.
- int vector_any (vector_t *v, int(*f)(void *))

  Return true if at least one element satisfies the predicate.
- void vector_fold (vector_t *v, void *acc, void(*fun)(void *, void *))

  Executes the function once for each array element.

6.2.1 Detailed Description

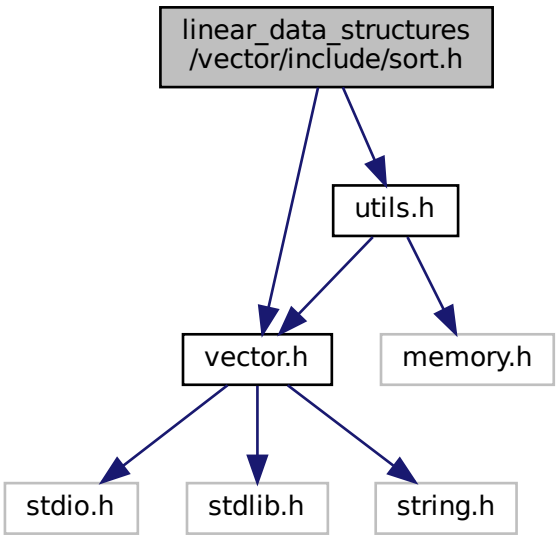
Header file for implement higher order functions.

6.3 linear_data_structures/vector/include/sort.h File Reference

Header file for implements sort algorithms.

```
#include "utils.h"
#include "vector.h"
```

Include dependency graph for sort.h:



Functions

- void vector_bubble_sort (vector_t *v, int(*cmp)(void *, void *))

  Implementation bubble sort algorithm.
- void vector_insertion_sort (vector_t *v, int(*cmp)(void *, void *))

  Implementation insertion sort algorithm.

6.3.1 Detailed Description

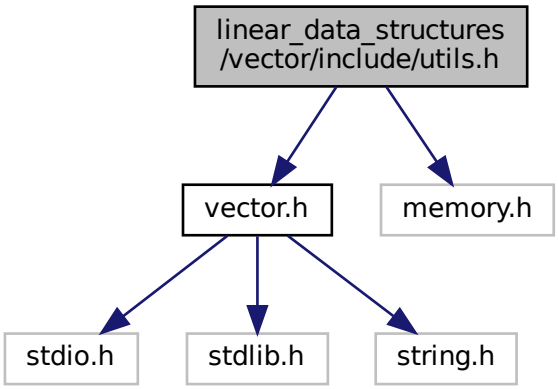
Header file for implements sort algorithms.

6.4 linear_data_structures/vector/include/utils.h File Reference

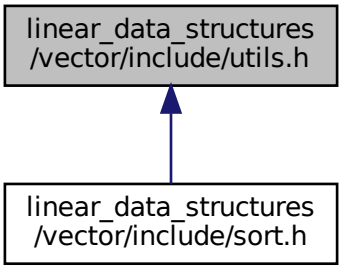
Header file for utils functions.

```
#include "vector.h"
#include <memory.h>
```

Include dependency graph for utils.h:



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



Functions

- void * **get_item** (vector_t *v, size_t index)
- void **vector_swap** (vector_t *v, size_t index1, size_t index2, size_t elem_size)

6.4.1 Detailed Description

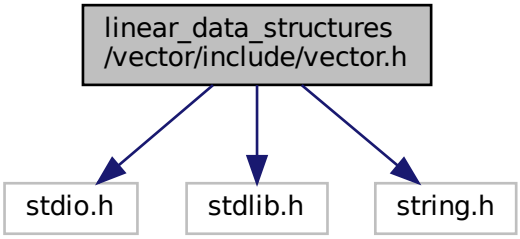
Header file for utils functions.

6.5 linear_data_structures/vector/include/vector.h File Reference

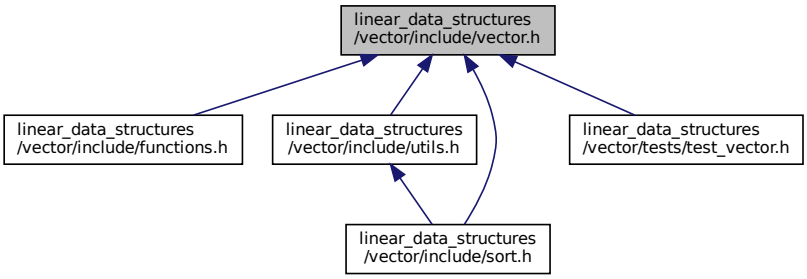
Header file for dynamic array (vector)

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

Include dependency graph for vector.h:



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



Classes

- struct vector_t

Macros

- #define **EQUAL** 0
- #define **LESS** 1
- #define **MORE** 2

Typedefs

- typedef struct vector_t **vector_t**

Functions

- void vector_init (vector_t *v, size_t capacity, size_t elem_size)

  Construct new vector data structure.
- void vector_null (vector_t *v)

  Implementation null object pattern.
- void vector_free (vector_t *v, void(*deleter)(void *))

  Free memory dynamic array.
- void vector_push_back (vector_t *v, void *elem)

  Adds an item to the end.
- void vector_insert_by_index (vector_t *v, size_t index, void *elem)

  Adds an item by index. If index > size then don't add.
- void vector_delete_by_value (vector_t *v, void *key, int(*cmp)(void *, void *), void(*deleter)(void *))

  Delete item by value, shifting all trailing elements left.
- void vector_delete_by_index (vector_t *v, size_t index, void(*deleter)(void *))

  Delete item at index, shifting all trailing elements left.
- void * vector_get (vector_t *v, size_t index)

  Returns item at given index.
- void vector_set (vector_t *v, size_t index, void *elem)

  Change element by index.
- size_t vector_size (vector_t *v)

  Returns size of vector (expressed in terms of elements).
- size_t vector_is_empty (vector_t *v)

  Returns true if vector is empty (expressed in terms of elements).
- long long vector_find (vector_t *v, void *key, int(*cmp)(void *, void *))

  Implementation linear search.
- long long vector_binary_search (vector_t *v, void *key, int(*cmp)(void *, void *))

  Implementation binary search.

6.5.1 Detailed Description

Header file for dynamic array (vector)

This file contains the definition of the data structure dynamic array