Binary search tree using C

0.1

Generated by Doxygen 1.8.17

1 Module Index 1

1.1 Modules . . . 1

2 Class Index 3

2.1 Class List . . . 3

3 File Index 5

3.1 File List . . . 5

4 Module Documentation 7

4.1 Binary_search_tree . . . 7

4.1.1 Detailed Description . . . 8

4.1.2 Typedef Documentation . . . 8

4.1.2.1 tree_t . . . 8

4.1.3 Function Documentation . . . 8

4.1.3.1 tree_add() . . . 9

4.1.3.2 tree_add_r() . . . 9

4.1.3.3 tree_delete() . . . 9

4.1.3.4 tree_find() . . . 10

4.1.3.5 tree_find_r() . . . 10

4.1.3.6 tree_height() . . . 10

4.1.3.7 tree_height_r() . . . 11

4.1.3.8 tree_in_order_travers() . . . 11

4.1.3.9 tree_init() . . . 11

4.1.3.10 tree_max() . . . 12

4.1.3.11 tree_max_r() . . . 12

4.1.3.12 tree_min() . . . 13

4.1.3.13 tree_min_r() . . . 13

4.1.3.14 tree_node_count() . . . 13

4.1.3.15 tree_node_count_r() . . . 14

4.1.3.16 tree_post_order_travers() . . . 14

4.1.3.17 tree_pre_order_travers() . . . 14

4.1.3.18 tree_predecessor() . . . 15

4.1.3.19 tree_successor() . . . 15

4.2 Stack . . . 16

4.2.1 Detailed Description . . . 16

4.2.2 Function Documentation . . . 16

4.2.2.1 stack_free() . . . 16

4.2.2.2 stack_init() . . . 16

4.2.2.3 stack_pop() . . . 17

4.2.2.4 stack_push() . . . 17

5 Class Documentation 19

5.1 stack_t Struct Reference . . . 19

5.2 tree_node_t Struct Reference 20
5.3 tree_t Struct Reference 20
5.3.1 Detailed Description 21

6 File Documentation **23**

6.1 include/binary_tree.h File Reference 23
6.1.1 Detailed Description 25

Index **27**

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Binary_search_tree 7
Stack 16

Chapter 2

Class Index

2.1 Class List

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

stack_t 19
tree_node_t 20
tree_t
Binary search tree data structure 20

Chapter 3

File Index

3.1 File List

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

include/binary_tree.h
Header file for Simple binary search tree This file contains the definition of the data structure binary search tree (naive implementation) 23
include/**stack.h** . . . . . **??**

Chapter 4

Module Documentation

4.1 Binary_search_tree

Item of binary search tree.

Classes

- struct tree_t

  binary search tree data structure

Typedefs

- typedef struct tree_t tree_t

  binary search tree data structure

Functions

- void tree_init (tree_t *table, int key, char *value)

  Init binary search tree data structure.
- void tree_add (tree_t *tree, int key, char *value)

  Add the given key and object to tree(iterative version).
- void tree_add_r (tree_t *tree, int key, char *value)

  Add the given key and object to tree(recursive version).
- const node_t * tree_find (tree_t *tree, int key)

  Finds key in tree(iterative version).
- const node_t * tree_find_r (tree_t *tree, int key)

  Finds key in tree(recursive version).
- const node_t * tree_min (const tree_t *tree)

  Returns node with min key(iterative version).
- const node_t * tree_min_r (const tree_t *tree)

  Returns node with min key(recursive version).
- const node_t * tree_max (const tree_t *tree)

  Returns node with max key(iterative version).

- const node_t * tree_max_r (const tree_t *tree)

  Returns node with max key(recursive version).
- size_t tree_height (const tree_t *tree)

  Returns the height of the tree in nodes, 0 if empty (iterative version).
- size_t tree_height_r (const tree_t *tree)

  Returns the height of the tree in nodes, 0 if empty (recursive version).
- size_t tree_node_count (const tree_t *tree)

  Returns the number of nodes stored in the tree(iterative version).
- size_t tree_node_count_r (const tree_t *tree)

  Returns the number of nodes stored in the tree(recursive version).
- const node_t * tree_successor (const tree_t *tree, int key)

  Returns the node that contains next key.
- const node_t * tree_predecessor (const tree_t *tree, int key)

  Returns the node contains previous key.
- void tree_pre_order_travers (tree_t *tree, void(*visit)(node_t *node, void *params), void *params)

  Visits nodes in preorder(root, left, right).
- void tree_in_order_travers (tree_t *tree, void(*visit)(node_t *node, void *params), void *params)

  Visits nodes in inorder(left, root, right).
- void tree_post_order_travers (tree_t *tree, void(*visit)(node_t *node, void *params), void *params)

  Visits nodes in postorder(left, right, root).
- void tree_delete (tree_t *tree)

  Deletes all nodes in tree, freeing their memory.

4.1.1 Detailed Description

Item of binary search tree.

Warning

This structure created only for educational goals

4.1.2 Typedef Documentation

4.1.2.1 tree_t

```
typedef struct tree_t tree_t
```

binary search tree data structure

Warning

This structure created only for educational goals

4.1.3 Function Documentation

4.1.3.1 tree_add()

```
void tree_add (
            tree_t * tree,
            int key,
            char * value )
```

Add the given key and object to tree(iterative version).

Parameters

tree	Pointer to binary search tree data structure.
key	Key for value.
value	Value by key.

4.1.3.2 tree_add_r()

```
void tree_add_r (
            tree_t * tree,
            int key,
            char * value )
```

Add the given key and object to tree(recursive version).

Parameters

tree	Pointer to binary search tree data structure.
key	Key for value.
value	Value by key.

4.1.3.3 tree_delete()

```
void tree_delete (
            tree_t * tree )
```

Deletes all nodes in tree, freeing their memory.

Parameters

tree	Pointer to binary search tree data structure.

4.1.3.4 tree_find()

```
const node_t* tree_find (
            tree_t * tree,
            int key )
```

Finds key in tree(iterative version).

Parameters

tree	Pointer to binary search tree data structure.
key	Key for find.

Returns

founded node or NULL if not found

4.1.3.5 tree_find_r()

```
const node_t* tree_find_r (
            tree_t * tree,
            int key )
```

Finds key in tree(recursive version).

Parameters

tree	Pointer to binary search tree data structure.
key	Key for find.

Returns

founded node or NULL if not found

4.1.3.6 tree_height()

```
size_t tree_height (
            const tree_t * tree )
```

Returns the height of the tree in nodes, 0 if empty (iterative version).

Parameters

tree	Pointer to binary search tree data structure.

Returns

height of three

4.1.3.7 tree_height_r()

```
size_t tree_height_r (
            const tree_t * tree )
```

Returns the height of the tree in nodes, 0 if empty (recursive version).

Parameters

tree	Pointer to binary search tree data structure.

Returns

height of three

4.1.3.8 tree_in_order_travers()

```
void tree_in_order_travers (
            tree_t * tree,
            void(*)(node_t *node, void *params) visit,
            void * params )
```

Visits nodes in inorder(left, root, right).

Parameters

tree	Pointer to binary search tree data structure.
visit	Function of working with nodes(first argument is node of tree, second argument is param(pointer to void)).
params	Params for visit function(additional parameters (if needed)).

Returns

ode that contains next key

4.1.3.9 tree_init()

```
void tree_init (
            tree_t * table,
```

```
            int key,
            char * value )
```

Init binary search tree data structure.

Parameters

tree	Pointer to binary search tree data structure.
key	Key of root node.
value	Value of root node.

4.1.3.10 tree_max()

```
const node_t* tree_max (
            const tree_t * tree )
```

Returns node with max key(iterative version).

Parameters

tree	Pointer to binary search tree data structure.

Returns

founded node or NULL if not found

4.1.3.11 tree_max_r()

```
const node_t* tree_max_r (
            const tree_t * tree )
```

Returns node with max key(recursive version).

Parameters

tree	Pointer to binary search tree data structure.

Returns

founded node or NULL if not found

4.1.3.12 tree_min()

```
const node_t* tree_min (
            const tree_t * tree )
```

Returns node with min key(iterative version).

Parameters

tree	Pointer to binary search tree data structure.

Returns

founded node or NULL if not found

4.1.3.13 tree_min_r()

```
const node_t* tree_min_r (
            const tree_t * tree )
```

Returns node with min key(recursive version).

Parameters

tree	Pointer to binary search tree data structure.

Returns

founded node or NULL if not found

4.1.3.14 tree_node_count()

```
size_t tree_node_count (
            const tree_t * tree )
```

Returns the number of nodes stored in the tree(iterative version).

Parameters

tree	Pointer to binary search tree data structure.

Returns

count of nodes

4.1.3.15 tree_node_count_r()

```
size_t tree_node_count_r (
            const tree_t * tree )
```

Returns the number of nodes stored in the tree(recursive version).

Parameters

tree	Pointer to binary search tree data structure.

Returns

count of nodes

4.1.3.16 tree_post_order_travers()

```
void tree_post_order_travers (
            tree_t * tree,
            void(*)(node_t *node, void *params) visit,
            void * params )
```

Visits nodes in postorder(left, right, root).

Parameters

tree	Pointer to binary search tree data structure.
visit	Function of working with nodes(first argument is node of tree, second argument is param(pointer to void)).
params	Params for visit function(additional parameters (if needed)).

Returns

ode that contains next key

4.1.3.17 tree_pre_order_travers()

```
void tree_pre_order_travers (
            tree_t * tree,
            void(*)(node_t *node, void *params) visit,
            void * params )
```

Visits nodes in preorder(root, left, right).

Parameters

tree	Pointer to binary search tree data structure.
visit	Function of working with nodes(first argument is node of tree, second argument is param(pointer to void)).
params	Params for visit function(additional parameters (if needed)).

Returns

ode that contains next key

4.1.3.18 tree_predecessor()

```
const node_t* tree_predecessor (
            const tree_t * tree,
            int key )
```

Returns the node contains previous key.

Parameters

tree	Pointer to binary search tree data structure.

Returns

ode that contains next key

4.1.3.19 tree_successor()

```
const node_t* tree_successor (
            const tree_t * tree,
            int key )
```

Returns the node that contains next key.

Parameters

tree	Pointer to binary search tree data structure.

Returns

ode that contains next key

4.2 Stack

Stack Implementation for DFS(using dynamic array)

Functions

- void stack_init (stack_t *stack, size_t capacity)

  Init stack data structure.
- void stack_push (stack_t *stack, node_t *node)

  Adds node in stack.
- node_t * stack_pop (stack_t *stack)

  Extract node from stack.
- void stack_free (stack_t *stack)

  Free memory in stack_item_t.

4.2.1 Detailed Description

Stack Implementation for DFS(using dynamic array)

Warning

This structure created only for educational goals

4.2.2 Function Documentation

4.2.2.1 stack_free()

```
void stack_free (
            stack_t * stack )
```

Free memory in stack_item_t.

Parameters

s	Pointer to stack.

4.2.2.2 stack_init()

```
void stack_init (
            stack_t * stack,
            size_t capacity )
```

Init stack data structure.

Parameters

stack	Pointer to stack data structure.
capacity	Capacity of stack

4.2.2.3 stack_pop()

```
node_t* stack_pop (
            stack_t * stack )
```

Extract node from stack.

Parameters

stack	Stack data structure.

Returns

node Element of stack.

4.2.2.4 stack_push()

```
void stack_push (
            stack_t * stack,
            node_t * node )
```

Adds node in stack.

Parameters

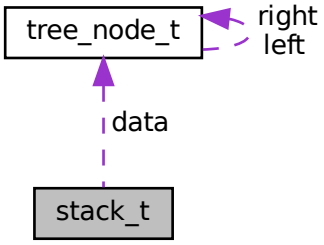
stack	Stack data structure.
node	Element for added in stack.

Chapter 5

Class Documentation

5.1 stack_t Struct Reference

Collaboration diagram for stack_t:



Public Attributes

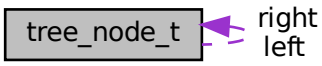
- node_t ** **data**
- size_t **count**
- size_t **capacity**

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

- include/stack.h

5.2 tree_node_t Struct Reference

Collaboration diagram for tree_node_t:



Public Attributes

- char * **value**
- struct tree_node_t * **left**
- struct tree_node_t * **right**
- int **key**

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

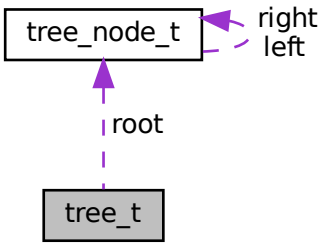
- include/binary_tree.h

5.3 tree_t Struct Reference

binary search tree data structure

`#include <binary_tree.h>`

Collaboration diagram for tree_t:



Public Attributes

- node_t * **root**
- size_t **count**

5.3.1 Detailed Description

binary search tree data structure

Warning

This structure created only for educational goals

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

- include/binary_tree.h

Chapter 6

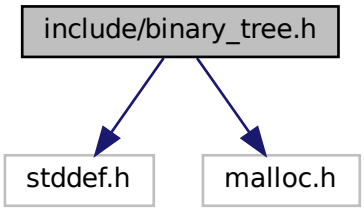
File Documentation

6.1 include/binary_tree.h File Reference

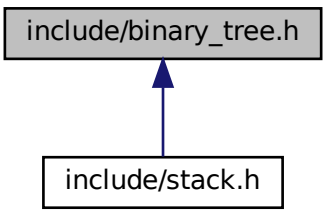
Header file for Simple binary search tree This file contains the definition of the data structure binary search tree (naive implementation)

```
#include <stddef.h>
#include <malloc.h>
```

Include dependency graph for binary_tree.h:



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



Classes

- struct tree_node_t
- struct tree_t

  binary search tree data structure

Typedefs

- typedef struct tree_node_t **node_t**
- typedef struct tree_t tree_t

  binary search tree data structure

Functions

- void tree_init (tree_t *table, int key, char *value)

  Init binary search tree data structure.
- void tree_add (tree_t *tree, int key, char *value)

  Add the given key and object to tree(iterative version).
- void tree_add_r (tree_t *tree, int key, char *value)

  Add the given key and object to tree(recursive version).
- const node_t * tree_find (tree_t *tree, int key)

  Finds key in tree(iterative version).
- const node_t * tree_find_r (tree_t *tree, int key)

  Finds key in tree(recursive version).
- const node_t * tree_min (const tree_t *tree)

  Returns node with min key(iterative version).
- const node_t * tree_min_r (const tree_t *tree)

  Returns node with min key(recursive version).
- const node_t * tree_max (const tree_t *tree)

  Returns node with max key(iterative version).
- const node_t * tree_max_r (const tree_t *tree)

  Returns node with max key(recursive version).
- size_t tree_height (const tree_t *tree)

  Returns the height of the tree in nodes, 0 if empty (iterative version).
- size_t tree_height_r (const tree_t *tree)

  Returns the height of the tree in nodes, 0 if empty (recursive version).
- size_t tree_node_count (const tree_t *tree)

  Returns the number of nodes stored in the tree(iterative version).
- size_t tree_node_count_r (const tree_t *tree)

  Returns the number of nodes stored in the tree(recursive version).
- const node_t * tree_successor (const tree_t *tree, int key)

  Returns the node that contains next key.
- const node_t * tree_predecessor (const tree_t *tree, int key)

  Returns the node contains previous key.
- void tree_pre_order_travers (tree_t *tree, void(*visit)(node_t *node, void *params), void *params)

  Visits nodes in preorder(root, left, right).
- void tree_in_order_travers (tree_t *tree, void(*visit)(node_t *node, void *params), void *params)

  Visits nodes in inorder(left, root, right).
- void tree_post_order_travers (tree_t *tree, void(*visit)(node_t *node, void *params), void *params)

  Visits nodes in postorder(left, right, root).
- void tree_delete (tree_t *tree)

  Deletes all nodes in tree, freeing their memory.

6.1.1 Detailed Description

Header file for Simple binary search tree This file contains the definition of the data structure binary search tree (naive implementation)

Index

Binary_search_tree, 7
    tree_add, 8
    tree_add_r, 9
    tree_delete, 9
    tree_find, 9
    tree_find_r, 10
    tree_height, 10
    tree_height_r, 11
    tree_in_order_travers, 11
    tree_init, 11
    tree_max, 12
    tree_max_r, 12
    tree_min, 12
    tree_min_r, 13
    tree_node_count, 13
    tree_node_count_r, 14
    tree_post_order_travers, 14
    tree_pre_order_travers, 14
    tree_predecessor, 15
    tree_successor, 15
    tree_t, 8

include/binary_tree.h, 23

Stack, 16
    stack_free, 16
    stack_init, 16
    stack_pop, 17
    stack_push, 17
stack_free
    Stack, 16
stack_init
    Stack, 16
stack_pop
    Stack, 17
stack_push
    Stack, 17
stack_t, 19

tree_add
    Binary_search_tree, 8
tree_add_r
    Binary_search_tree, 9
tree_delete
    Binary_search_tree, 9
tree_find
    Binary_search_tree, 9
tree_find_r
    Binary_search_tree, 10
tree_height
    Binary_search_tree, 10
tree_height_r
    Binary_search_tree, 11
tree_in_order_travers
    Binary_search_tree, 11
tree_init
    Binary_search_tree, 11
tree_max
    Binary_search_tree, 12
tree_max_r
    Binary_search_tree, 12
tree_min
    Binary_search_tree, 12
tree_min_r
    Binary_search_tree, 13
tree_node_count
    Binary_search_tree, 13
tree_node_count_r
    Binary_search_tree, 14
tree_node_t, 20
tree_post_order_travers
    Binary_search_tree, 14
tree_pre_order_travers
    Binary_search_tree, 14
tree_predecessor
    Binary_search_tree, 15
tree_successor
    Binary_search_tree, 15
tree_t, 20
    Binary_search_tree, 8