BinarySearchTree

Generated by Doxygen 1.8.11

Contents

1	Data	Structure Index	1	
	1.1	Data Structures	. 1	
2	File	ndex	2	
	2.1	File List	. 2	
3	Data	Structure Documentation	2	
	3.1	BST_t Struct Reference	. 2	
		3.1.1 Detailed Description	. 2	
		3.1.2 Field Documentation	. 3	
	3.2	BSTnode_t Struct Reference	. 3	
		3.2.1 Detailed Description	. 4	
		3.2.2 Field Documentation	. 4	
4	File	ocumentation	5	
	4.1	bst.c File Reference	. 5	
		4.1.1 Function Documentation	. 5	
	4.2	bst.c	. 9	
	4.3	bst.h File Reference	. 11	
		4.3.1 Detailed Description	. 12	
		4.3.2 Typedef Documentation	. 13	
		4.3.3 Function Documentation	. 13	
	4.4	bst.h	. 16	
Index				
1	Da	a Structure Index		
1.1	Da	a Structures		
Llare are the data atmentures with heist descriptions.				
Here are the data structures with brief descriptions:				
	BST_t			

BSTnode_t 3

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

bst.c 5

bst.h

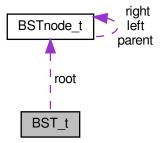
Binary search tree definition and basic operations

3 Data Structure Documentation

3.1 BST_t Struct Reference

#include <bst.h>

Collaboration diagram for BST_t:



Data Fields

- size_t width
- BSTnode_t * root
- int count
- int height
- int(* compare)(const void *, const void *)

3.1.1 Detailed Description

Definition at line 22 of file bst.h.

```
3.1.2 Field Documentation

3.1.2.1 int(* compare) (const void *, const void *)

Definition at line 27 of file bst.h.

3.1.2.2 int count

count element amount

Definition at line 25 of file bst.h.

3.1.2.3 int height

tree height

Definition at line 26 of file bst.h.
```

root node

Definition at line 24 of file bst.h.

3.1.2.4 BSTnode_t* root

3.1.2.5 size_t width

element size (in bytes)

Definition at line 23 of file bst.h.

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

• bst.h

3.2 BSTnode_t Struct Reference

#include <bst.h>

Collaboration diagram for BSTnode_t:



Data Fields

struct BSTnode_t * parent

```
• struct BSTnode_t * left
    • struct BSTnode_t * right
    void * data
3.2.1 Detailed Description
Definition at line 15 of file bst.h.
3.2.2 Field Documentation
3.2.2.1 void* data
data pointer
Definition at line 19 of file bst.h.
3.2.2.2 struct BSTnode_t* left
left child
Definition at line 17 of file bst.h.
3.2.2.3 struct BSTnode_t* parent
parent node
Definition at line 16 of file bst.h.
3.2.2.4 struct BSTnode_t* right
right child
```

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

• bst.h

Definition at line 18 of file bst.h.

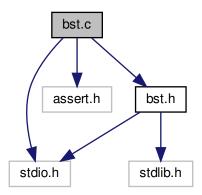
4 File Documentation 5

4 File Documentation

4.1 bst.c File Reference

```
#include <stdio.h>
#include <assert.h>
#include "bst.h"
```

Include dependency graph for bst.c:



Functions

```
• BST_t * bst_create (size_t width, int(*compare)(const void *, const void *))
```

- void bst_destruct (BST_t *bst)
- BSTnode_t * bst_search (BST_t *bst, void *data)
- BSTnode_t * bst_iterative_search (BST_t *bst, void *data)
- BSTnode_t * bst_min (BSTnode_t *node)
- BSTnode_t * bst_max (BSTnode_t *node)
- BSTnode_t * bst_succ (BST_t *bst, BSTnode_t *node)
- BSTnode_t * bst_pred (BST_t *bst, BSTnode_t *node)
- void bst_insert (BST_t *bst, void *data)
- void bst_delete (BST_t *bst, BSTnode_t *node)
- void bst_inorder_walk_int (BSTnode_t *node)

4.1.1 Function Documentation

4.1.1.1 BST t* bst_create (size_t width, int(*)(const void *, const void *) compare)

Definition at line 6 of file bst.c.

```
00006
00007
           BST_t *bst = malloc(sizeof(BST_t));
           assert(bst);
bst->root = NULL;
bst->count = 0;
00008
00009
00010
00011
           bst->height = 0;
00012
           bst->width = width;
00013
           bst->compare = compare;
00014
            return bst;
00015 }
```

4.1.1.2 void bst_delete (BST_t * bst, BSTnode_t * node)

Definition at line 164 of file bst.c.

```
00164
00165
             if (!node->left) {
00166
                 bst_transplant(bst, node, node->right);
00167
             } else if (!node->right){
00168
                 bst_transplant(bst, node, node->left);
00169
            } else {
                 BSTnode_t* y = bst_min(node->right);
if (y->parent != node) {
00170
00171
                       ty >parent - Node)
bst_transplant (bst, y, y->right);
y->right = node->right;
y->right->parent = y;
00172
00173
00174
00175
00176
                 bst_transplant(bst, node, y);
y->left = node->left;
00177
00178
                  y->left->parent = y;
00179
00180 }
```

Here is the call graph for this function:



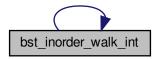
4.1.1.3 void bst_destruct (BST_t * bst)

Definition at line 28 of file bst.c.

4.1.1.4 void bst_inorder_walk_int (BSTnode_t * node)

Definition at line 184 of file bst.c.

Here is the call graph for this function:



4.1 bst.c File Reference 7

```
4.1.1.5 void bst_insert ( BST_t * bst, void * data )
```

Definition at line 140 of file bst.c.

4.1.1.6 BSTnode_t* bst_iterative_search (BST_t * bst, void * data)

Definition at line 50 of file bst.c.

```
00050
00051
         BSTnode_t *node = bst->root;
00052
         while (node && !bst->compare(data, node->data)) {
00053
00054
           if (bst->compare(data, node->data) < 0) {</pre>
00055
                 node = node->left;
00056
00057
             } else {
00058
                 node = node->right;
00059
             }
00060
         }
00061
00062
         return node;
00063 }
```

4.1.1.7 BSTnode_t* bst_max (BSTnode_t * node)

Definition at line 74 of file bst.c.

4.1.1.8 BSTnode_t* bst_min(BSTnode_t * node)

Definition at line 65 of file bst.c.

```
4.1.1.9 BSTnode_t* bst_pred ( BST_t * bst, BSTnode_t * node )
```

Definition at line 97 of file bst.c.

```
00097
00098
         if (node->left) {
            return bst_max(node->left);
00099
00100
        }
00102
        BSTnode_t *x = node->parent;
00103
00104
        while (x && x->left && bst->compare(node->data, x->left->
00106
            x = x->parent;
00107
00108
00109
        return x;
00110 }
```

Here is the call graph for this function:



```
4.1.1.10 BSTnode_t* bst_search ( BST_t * bst, void * data )
```

Definition at line 46 of file bst.c.

4.1.1.11 BSTnode_t* bst_succ (BST_t * bst, BSTnode_t * node)

Definition at line 82 of file bst.c.

```
00082
00083
          if (node->right) {
00084
             return bst_min(node->right);
00085
00086
00087
         BSTnode_t *x = node->parent;
00088
00089
        while (x && x->right && bst->compare(node->data, x->right->
     data) == 0) {
00090
             node = x;
00091
00092
             x = x->parent;
          }
00093
00094
          return x;
00095 }
```

4.2 bst.c 9

Here is the call graph for this function:



4.2 bst.c

```
00001 #include <stdio.h>
00002 #include <assert.h>
00003
00004 #include "bst.h"
00005
00006 BST_t *bst_create(size_t width, int (*compare)(const void *, const void *)) {
00007
         BST_t *bst = malloc(sizeof(BST_t));
80000
          assert(bst);
          bst->root = NULL;
bst->count = 0;
00009
00010
00011
          bst->height = 0;
          bst->width = width;
00012
00013
          bst->compare = compare;
00014
          return bst;
00015 }
00016
00017 static void bst_destruct_node(BSTnode_t *node) {
00018
         if (!node) {
00019
              return;
00020
00021
00022
          free(node->data);
00023
          bst_destruct_node(node->left);
00024
          bst_destruct_node(node->right);
00025
          free (node);
00026 }
00027
00028 void bst_destruct(BST_t *bst) {
00029
         bst_destruct_node(bst->root);
00030
          free (bst);
00031 }
00032
00033 static BSTnode_t *bst_search_node(BSTnode_t *node, void *data, int (*compare)(const void
     *, const void *)) {
00034
         if (node || !compare(data, node->data)) {
00035
              return node;
00036
00037
00038
          if (compare(data, node->data) < 0) {</pre>
00039
             return bst_search_node(node->left, data, compare);
00040
00041
          } else {
00042
             return bst_search_node(node->right, data, compare);
00043
00044 }
00045
00046 BSTnode_t *bst_search(BST_t *bst, void *data) {
00047
          return bst_search_node(bst->root, data, bst->compare);
00048 }
00049
00050 BSTnode_t *bst_iterative_search(BST_t *bst, void *data) {
00051
         BSTnode_t *node = bst->root;
00052
00053
          while (node && !bst->compare(data, node->data)) {
             if (bst->compare(data, node->data) < 0) {
00054
00055
                 node = node->left;
00056
00057
              } else {
00058
                 node = node->right;
00059
              }
00060
          }
00061
00062
          return node;
```

```
00063 }
00064
00065 BSTnode_t *bst_min(BSTnode_t *node) {
00066
         while (node->left) {
00067
             node = node->left;
00068
00069
00070
          return node;
00071 }
00072
00073
00074 BSTnode_t *bst_max(BSTnode_t *node) {
00075    while (node->right) {
00076
            node = node->right;
00077
00078
00079
          return node;
00080 }
00081
00082 BSTnode_t *bst_succ(BST_t *bst, BSTnode_t *node) {
00083
         if (node->right) {
00084
             return bst_min(node->right);
00085
00086
00087
          BSTnode_t *x = node->parent;
00088
00089
         while (x && x->right && bst->compare(node->data, x->right->
     data) == 0) {
00090
             node = x;
00091
             x = x->parent;
00092
          }
00093
00094
          return x;
00095 }
00096
00097 BSTnode_t *bst_pred(BST_t *bst, BSTnode_t *node) {
00098
         if (node->left) {
              return bst_max(node->left);
00100
00101
00102
          BSTnode_t *x = node->parent;
00103
          while (x && x->left && bst->compare(node->data, x->left->
00104
     data) == 0) {
00105
           node = x;
00106
             x = x->parent;
00107
          }
00108
00109
          return x:
00110 }
00111
00112 //TODO:update height and count
00113 void static bst_insert_node(BST_t *bst, BSTnode_t *node) {
00114
          BSTnode_t *x = bst->root, *y = NULL;
00115
00116
          while (x) {
             y = x;
00118
00119
             if (bst->compare(node->data, x->data) < 0) {</pre>
00120
                  x = x \rightarrow left;
00121
00122
              } else {
00123
                  x = x -> right;
00124
00125
          }
00126
00127
          node->parent = y;
00128
00129
          if (!y) {
00130
             bst->root = node;
00131
00132
          } else if (bst->compare(node->data, y->data) < 0) {</pre>
00133
             y->left = node;
00134
00135
          } else {
00136
             y->right = node;
00137
00138 }
00139
00140 void bst_insert(BST_t* bst, void* data){
00141
         BSTnode_t* node = malloc(sizeof(BSTnode_t));
          node->data = data;
00142
00143
          node->parent = NULL;
00144
          node->left = NULL;
00145
          node->right = NULL;
00146
          bst_insert_node(bst, node);
00147 }
```

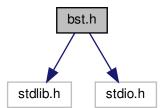
4.3 bst.h File Reference 11

```
00148
00149 //TODO:update height and count
00150 static void bst_transplant(BST_t* bst, BSTnode_t* u, BSTnode_t* v){
00151
          if (!u->parent) {
          bst->root = v;
} else if (u == u->parent->left) {
  u->parent->left = v;
00152
00153
00154
00155
          } else {
00156
             u->parent->right = v;
00157
           if (v) {
00158
00159
               v->parent = u->parent;
00160
           }
00161 }
00162
00163 //TODO:update height and count
00164 void bst_delete(BST_t* bst, BSTnode_t* node){
         if (!node->left) {
00165
               bst_transplant(bst, node, node->right);
00166
00167
          } else if (!node->right) {
00168
               bst_transplant(bst, node, node->left);
00169
         } else {
               BSTnode_t* y = bst_min(node->right);
if (y->parent != node) {
00170
00171
                    bst_transplant(bst, y, y->right);
y->right = node->right;
00172
00173
00174
                    y->right->parent = y;
00175
00176
               bst_transplant(bst, node, y);
               y->left = node->left;
y->left->parent = y;
00177
00178
00179
           }
00180 }
00181
00182
00183 //TODO:generalize this function
00184 void bst_inorder_walk_int(BSTnode_t* node){
00185 if(!node) return;
00186
           bst_inorder_walk_int(node->left);
00187
           printf("%d\n", *((int*)(node->data)));
00188
           bst_inorder_walk_int(node->right);
00189 }
00190
```

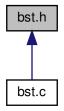
4.3 bst.h File Reference

binary search tree definition and basic operations

```
#include <stdlib.h>
#include <stdio.h>
Include dependency graph for bst.h:
```



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



Data Structures

- struct BSTnode_t
- struct BST_t

Typedefs

- typedef struct BSTnode_t BSTnode_t
- typedef struct BST_t BST_t

Functions

- BST_t * bst_create (size_t width, int(*compare)(const void *, const void *))
- void bst_destruct (BST_t *bst)
- void bst_insert (BST_t *bst, void *data)
- void bst_delete (BST_t *bst, BSTnode_t *node)
- BSTnode_t * bst_search (BST_t *bst, void *data)
- BSTnode_t * bst_iterative_search (BST_t *bst, void *data)
- BSTnode_t * bst_max (BSTnode_t *node)
- BSTnode_t * bst_min (BSTnode_t *node)
- BSTnode_t * bst_succ (BST_t *bst, BSTnode_t *node)
- BSTnode_t * bst_pred (BST_t *bst, BSTnode_t *node)
- void bst_inorder_walk_int (BSTnode_t *node)

4.3.1 Detailed Description

binary search tree definition and basic operations

Author

Firmin MARTIN

Version

0.1

Date

16/03/2018

Definition in file bst.h.

4.3 bst.h File Reference 13

- 4.3.2 Typedef Documentation
- 4.3.2.1 typedef struct BST_t BST_t
- 4.3.2.2 typedef struct BSTnode_t BSTnode_t
- 4.3.3 Function Documentation
- 4.3.3.1 BST_t* bst_create (size_t width, int(*)(const void *, const void *) compare)

Definition at line 6 of file bst.c.

```
00006
00007    BST_t *bst = malloc(sizeof(BST_t));
00008    assert(bst);
00009    bst->root = NULL;
00010    bst->count = 0;
00011    bst->height = 0;
00012    bst->width = width;
00013    bst->compare = compare;
00014    return bst;
00015 }
```

4.3.3.2 void bst_delete (BST_t * bst, BSTnode_t * node)

Definition at line 164 of file bst.c.

```
00164
00165
            if (!node->left) {
            bst_transplant(bst, node, node->right);
} else if (!node->right){
00166
00167
00168
                 bst_transplant(bst, node, node->left);
00169
                BSTnode_t* y = bst_min(node->right);
if (y->parent != node) {
00170
00171
00172
                      bst_transplant(bst, y, y->right);
y->right = node->right;
00174
                      y->right->parent = y;
00175
00176
                 bst_transplant(bst, node, y);
00177
                  y->left = node->left;
y->left->parent = y;
00178
00179
             }
00180 }
```

Here is the call graph for this function:



```
4.3.3.3 void bst_destruct ( BST_t * bst )
```

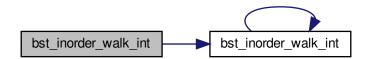
Definition at line 28 of file bst.c.

4.3.3.4 void bst_inorder_walk_int (BSTnode_t * node)

Definition at line 184 of file bst.c.

```
00184
00185     if(!node) return;
00186     bst_inorder_walk_int(node->left);
00187     printf("%d\n", *((int*)(node->data)));
00188     bst_inorder_walk_int(node->right);
00189 }
```

Here is the call graph for this function:



4.3.3.5 void bst_insert (BST_t * bst, void * data)

Definition at line 140 of file bst.c.

4.3.3.6 BSTnode t* bst_iterative_search (BST_t * bst, void * data)

Definition at line 50 of file bst.c.

```
00050
00051
          BSTnode_t *node = bst->root;
00052
00053
          while (node && !bst->compare(data, node->data)) {
00054
              if (bst->compare(data, node->data) < 0) {</pre>
00055
                  node = node->left;
00056
00057
              } else {
00058
                  node = node->right;
00059
00060
          }
00061
00062
          return node;
00063 }
```

4.3 bst.h File Reference 15

```
4.3.3.7 BSTnode_t* bst_max ( BSTnode_t * node )
```

Definition at line 74 of file bst.c.

4.3.3.8 BSTnode_t* bst_min(BSTnode_t * node)

Definition at line 65 of file bst.c.

4.3.3.9 BSTnode_t* bst_pred (BST_t * bst, BSTnode_t * node)

Definition at line 97 of file bst.c.

```
{
00098
         if (node->left) {
         return bst_max(node->left);
}
00099
00100
00101
00102
         BSTnode_t *x = node->parent;
00103
00104
        while (x && x->left && bst->compare(node->data, x->left->
     data) == 0) {
00105
            node = x;
            x = x->parent;
00106
00107
00108
00109
         return x;
00110 }
```

Here is the call graph for this function:



```
4.3.3.10 BSTnode_t* bst_search ( BST_t * bst, void * data )
```

Definition at line 46 of file bst.c.

4.3.3.11 BSTnode_t * bst_succ (BST_t * bst, BSTnode_t * node)

Definition at line 82 of file bst.c.

```
00082
00083
          if (node->right) {
00084
              return bst min(node->right);
00085
00086
00087
          BSTnode_t *x = node->parent;
00088
          while (x && x->right && bst->compare(node->data, x->right->
00089
     data) == 0) {
00090
             node = x;
00091
             x = x->parent;
00092
          }
00093
00094
          return x;
00095 }
```

Here is the call graph for this function:



4.4 bst.h

```
00001 #ifndef BST_H
00002 #define BST_H
00003
00012 #include <stdlib.h>
00013 #include <stdio.h>
00014
00015 typedef struct BSTnode_t {
        struct BSTnode_t* parent;
struct BSTnode_t* left;
00016
00017
           struct BSTnode_t* right;
00018
00019
           void* data:
00020 }BSTnode_t;
00021
00022 typedef struct BST_t {
00023
          size_t width;
00024
           BSTnode_t* root;
00025
           int count;
int height;
00026
           int (*compare)(const void *, const void *);
00028 }BST_t;
00029
00030 BST_t* bst_create(size_t width, int (*compare)(const void *, const void *));
00031 void bst_destruct(BST_t *bst);
00032 void bst_insert(BST_t *bst, void *data);
00033 void bst_delete(BST_t* bst, BSTnode_t* node);
00034 BSTnode_t* bst_search(BST_t *bst, void *data);
00035 BSTnode_t* bst_iterative_search(BST_t *bst, void *
      data);
00036 BSTnode_t* bst_max(BSTnode_t *node);
00037 BSTnode_t* bst_min(BSTnode_t *node);
00038 BSTnode_t* bst_succ(BST_t *bst, BSTnode_t *node);
00039 BSTnode_t* bst_pred(BST_t *bst, BSTnode_t *node);
00040 void bst_inorder_walk_int(BSTnode_t* node);
00041
00042 #endif
```

Index

BST_t, 2	bst.c, 7
bst.h, 13	bst.h, 14
compare, 3	bst_max
count, 3	bst.c, 7
height, 3	bst.h, 14
root, 3	bst_min
width, 3	bst.c, 7
BSTnode_t, 3	bst.h, 15
bst.h, 13	bst_pred
data, 4	bst.c, 7
left, 4	bst.h, 15
parent, 4	bst_search
right, 4	bst.c, 8
bst.c, 5	bst.h, 15
bst_create, 5	bst_succ
bst_delete, 5	bst.c, 8
bst_destruct, 6	bst.h, 15
bst_inorder_walk_int, 6	
bst_insert, 6	compare
bst_iterative_search, 7	BST_t, 3
bst_max, 7	count
bst_min, 7	BST_t, 3
bst pred, 7	data
bst_search, 8	data
bst_succ, 8	BSTnode_t, 4
bst.h, 11	height
BST_t, 13	BST_t, 3
BSTnode_t, 13	BO1_t, 0
bst_create, 13	left
bst_delete, 13	BSTnode_t, 4
bst_destruct, 13	
bst_inorder_walk_int, 14	parent
bst_insert, 14	BSTnode_t, 4
bst iterative search, 14	
bst max, 14	right
bst_min, 15	BSTnode_t, 4
bst_pred, 15	root
bst_search, 15	BST_t, 3
bst_succ, 15	!
bst create	width
bst.c, 5	BST_t, 3
bst.h, 13	
bst delete	
bst.c, 5	
bst.h, 13	
bst destruct	
bst.c, 6	
bst.h, 13	
bst inorder walk int	
bst_inorder_walk_int bst.c. 6	
bst.c, 6	
bst.c, 6 bst.h, 14	
bst.c, 6 bst.h, 14 bst_insert	
bst.c, 6 bst.h, 14 bst_insert bst.c, 6	
bst.c, 6 bst.h, 14 bst_insert	