Reference Manual

Generated by Doxygen 1.7.4

Wed Mar 14 2012 18:59:50

Contents

Chapter 1

mkavl: Multi-Key AVL Trees

Author

Matthew J. Miller (matt@matthewjmiller.net)

Date

2011

1.1 Introduction

The idea behind multi-key AVL trees is that a single item can be keyed multiple ways to different allow *O(lg N)* lookups. The simplest example would be if you have items that have two separate key fields. For example, if you have an employee database where each employee has a unique ID and phone number. The multi-key AVL essentially manages two separate AVLs under the covers for the same employee data, one keyed by ID and one by phone number, so an employee can be looked up efficiently by either field.

When the mkavl tree is created, *M* different comparison functions are given. When mkavl_add is called, the item is inserted in *M* different AVLs, with all *M* AVL nodes pointing to the same data item.

The only similar data structure I could find was the multi_index library in C++'s Boost.

The backend AVL implementation comes from Ben Plaff's open source AVL library.

In essence, this adds two things the backend AVL implementation. First, it more transparently handles a single data item being added to multiple AVL trees (each keyed differently). Second, it provides for greater than and less than lookups for keys not in the tree. E.g., if you wanted to find the first ID greater than X, you can do this even if X itself doesn't exist in the AVL.

For the unit test of this library, see test_mkavl.c. For example usage, see employee_example.c and malloc_example.c.

1.2 Example

A more complex example is, say you want to be able to query your employee database to find all employees with a given last name without walking the entire database. Let's say the employee ID is the primary key. You could create an mkavl tree with two comparison function keyed in the following manner:

1. `Key1:` `<ID>`
2. `Key2:` `<LastName | ID>`

Knowing the zero is the minimum ID value, you could lookup the first employee with the last name "Smith" in *O(lg N)* time by doing a greater than or equal to lookup on the key `<"Smith" | 0>`. If the record returned doesn't have the the LastName "Smith", then there are no matching records. Otherwise, you can continue iterating through all the "Smith" records doing greater than lookups until you hit NULL or a non-"Smith" record.

1.3 Usage

Just run `make all` to build the dynamic and shared libraries in lib/. Use "-lmkavl" to link the library. Running `make clean` will remove generated files (e.g., .o).

1.4 GNU General Public License

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

Class Documentation

4.1 employee_ctx_st_ Struct Reference

Public Attributes

- uint32_t nodes_walked
- uint32_t match_cnt

4.1.1 Detailed Description

The context associated with the employee AVLs.

Definition at line 190 of file employee_example.c.

4.1.2 Member Data Documentation

4.1.2.1 uint32_t employee_ctx_st_::match_cnt

Counter for the number of matches found for a given test

Definition at line 194 of file employee_example.c.

4.1.2.2 uint32_t employee_ctx_st_::nodes_walked

Counter for the number of nodes walked for a given test

Definition at line 192 of file employee_example.c.

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

- examples/employee_example.c

4.2 employee_example_input_st_ Struct Reference

Public Attributes

- const employee_example_opts_st * opts
- mkavl_tree_handle tree_h

4.2.1 Detailed Description

The input structure to pass test parameters to functions.

Definition at line 180 of file employee_example.c.

4.2.2 Member Data Documentation

4.2.2.1 const employee_example_opts_st* employee_example_input_st_::opts

The input options for the run

Definition at line 182 of file employee_example.c.

4.2.2.2 mkavl_tree_handle employee_example_input_st_::tree_h

The tree for the run

Definition at line 184 of file employee_example.c.

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

- examples/employee_example.c

4.3 employee_example_opts_st_ Struct Reference

Public Attributes

- uint32_t employee_cnt
- uint32_t run_cnt
- uint32_t seed
- uint8_t verbosity
- employee_dist_e last_name_dist
- double zipf_alpha

4.3.1 Detailed Description

State for the current test execution.

Definition at line 114 of file employee_example.c.

4.3.2 Member Data Documentation

4.3.2.1 uint32_t employee_example_opts_st_::employee_cnt

The number of employees in the DB

Definition at line 116 of file employee_example.c.

4.3.2.2 employee_dist_e employee_example_opts_st_::last_name_dist

The distribution function to use for last names

Definition at line 124 of file employee_example.c.

4.3.2.3 uint32_t employee_example_opts_st_::run_cnt

The number of separate runs to do

Definition at line 118 of file employee_example.c.

4.3.2.4 uint32_t employee_example_opts_st_::seed

The RNG seed for the first run

Definition at line 120 of file employee_example.c.

4.3.2.5 uint8_t employee_example_opts_st_::verbosity

The verbosity level for the test

Definition at line 122 of file employee_example.c.

4.3.2.6 double employee_example_opts_st_::zipf_alpha

The alpha value to parameterize a Zipf distribution

Definition at line 126 of file employee_example.c.

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

- examples/employee_example.c

4.4 employee_obj_st_ Struct Reference

Public Attributes

- uint32_t id

- char first_name [MAX_NAME_LEN]
- char last_name [MAX_NAME_LEN]

4.4.1 Detailed Description

The data stored for employees.

Definition at line 168 of file employee_example.c.

4.4.2 Member Data Documentation

4.4.2.1 char employee_obj_st_::first_name[MAX_NAME_LEN]

First name

Definition at line 172 of file employee_example.c.

4.4.2.2 uint32_t employee_obj_st_::id

Unique ID for the employee

Definition at line 170 of file employee_example.c.

4.4.2.3 char employee_obj_st_::last_name[MAX_NAME_LEN]

Last name

Definition at line 174 of file employee_example.c.

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

- examples/employee_example.c

4.5 employee_walk_ctx_st_ Struct Reference

Public Attributes

- char lookup_last_name [MAX_NAME_LEN]

4.5.1 Detailed Description

Context for the walk of the employee AVLs.

Definition at line 200 of file employee_example.c.

4.5.2 Member Data Documentation

4.5.2.1 char employee_walk_ctx_st_::lookup_last_name[MAX_NAME_LEN]

Last name for which the walk is being done

Definition at line 202 of file employee_example.c.

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

- examples/employee_example.c

4.6 malloc_example_input_st_ Struct Reference

Public Attributes

- const malloc_example_opts_st * opts
- mkavl_tree_handle tree_h

4.6.1 Detailed Description

The input structure to pass test parameters to functions.

Definition at line 141 of file malloc_example.c.

4.6.2 Member Data Documentation

4.6.2.1 const malloc_example_opts_st* malloc_example_input_st_::opts

The input options for the run

Definition at line 143 of file malloc_example.c.

4.6.2.2 mkavl_tree_handle malloc_example_input_st_::tree_h

The tree for the run

Definition at line 145 of file malloc_example.c.

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

- examples/malloc_example.c

4.7 malloc_example_opts_st_ Struct Reference

Public Attributes

- uint32_t malloc_cnt
- size_t memory_size
- uint32_t run_cnt
- uint32_t seed
- uint8_t verbosity
- malloc_pattern_e pattern

4.7.1 Detailed Description

State for the current test execution.

Definition at line 108 of file malloc_example.c.

4.7.2 Member Data Documentation

4.7.2.1 uint32_t malloc_example_opts_st_::malloc_cnt

The max number of allocations at any given time

Definition at line 110 of file malloc_example.c.

4.7.2.2 size_t malloc_example_opts_st_::memory_size

The size of the memory

Definition at line 112 of file malloc_example.c.

4.7.2.3 malloc_pattern_e malloc_example_opts_st_::pattern

The allocation pattern to use

Definition at line 120 of file malloc_example.c.

4.7.2.4 uint32_t malloc_example_opts_st_::run_cnt

The number of separate runs to do

Definition at line 114 of file malloc_example.c.

4.7.2.5 uint32_t malloc_example_opts_st_::seed

The RNG seed for the first run

Definition at line 116 of file malloc_example.c.

4.7.2.6 uint8_t malloc_example_opts_st_::verbosity

The verbosity level for the test

Definition at line 118 of file malloc_example.c.

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

- examples/malloc_example.c

4.8 memblock_ctx_st_ Struct Reference

Public Attributes

- uint32_t nodes_walked

4.8.1 Detailed Description

The context associated with the memblock AVLs.

Definition at line 151 of file malloc_example.c.

4.8.2 Member Data Documentation

4.8.2.1 uint32_t memblock_ctx_st_::nodes_walked

Counter for the number of nodes walked for a given test

Definition at line 153 of file malloc_example.c.

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

- examples/malloc_example.c

4.9 memblock_obj_st_ Struct Reference

Public Attributes

- void * start_addr
- size_t byte_cnt
- bool is_allocated

4.9.1 Detailed Description

The data for a free/allocated memory block.

Definition at line 126 of file malloc_example.c.

4.9.2 Member Data Documentation

4.9.2.1 size_t memblock_obj_st_::byte_cnt

The byte count for how big the block is

Definition at line 133 of file malloc_example.c.

4.9.2.2 bool memblock_obj_st_::is_allocated

Whether the block is allocated or free

Definition at line 135 of file malloc_example.c.

4.9.2.3 void* memblock_obj_st_::start_addr

Starting address for the memory (this is obviously unique for each memory block).

Definition at line 131 of file malloc_example.c.

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

- examples/malloc_example.c

4.10 mkavl_allocator_st_ Struct Reference

`#include <mkavl.h>`

Public Attributes

- mkavl_malloc_fn malloc_fn
- mkavl_free_fn free_fn

4.10.1 Detailed Description

Specifies the allocator functions for an AVL tree.

Definition at line 174 of file mkavl.h.

4.10.2 Member Data Documentation

4.10.2.1 mkavl_free_fn mkavl_allocator_st_::free_fn

The freeing function

Definition at line 178 of file mkavl.h.

4.10.2.2 mkavl_malloc_fn mkavl_allocator_st_::malloc_fn

The allocating function

Definition at line 176 of file mkavl.h.

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

- mkavl.h

4.11 mkavl_allocator_wrapper_st_ Struct Reference

Public Attributes

- struct libavl_allocator avl_allocator
- uint32_t magic
- mkavl_allocator_st mkavl_allocator
- mkavl_tree_handle tree_h

4.11.1 Detailed Description

A wrapper structure to map the mkavl_allocator to the avl_allocator.

Definition at line 86 of file mkavl.c.

4.11.2 Member Data Documentation

4.11.2.1 struct libavl_allocator mkavl_allocator_wrapper_st_::avl_allocator

The libavl allocator. Must be the first member for use in casting.

Definition at line 90 of file mkavl.c.

4.11.2.2 uint32_t mkavl_allocator_wrapper_st_::magic

Used for sanity checks

Definition at line 92 of file mkavl.c.

4.11.2.3 mkavl_allocator_st mkavl_allocator_wrapper_st_::mkavl_allocator

The mkavl allocator

Definition at line 94 of file mkavl.c.

4.11.2.4 mkavl_tree_handle mkavl_allocator_wrapper_st_::tree_h

The tree using the allocator

Definition at line 96 of file mkavl.c.

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

- mkavl.c

4.12 mkavl_avl_ctx_st_ Struct Reference

Public Attributes

- uint32_t magic
- mkavl_tree_handle tree_h
- size_t key_idx

4.12.1 Detailed Description

The internal context data for AVL callbacks.

Definition at line 62 of file mkavl.c.

4.12.2 Member Data Documentation

4.12.2.1 size_t mkavl_avl_ctx_st_::key_idx

The index in the mkavl tree of the AVL tree for the callback

Definition at line 68 of file mkavl.c.

4.12.2.2 uint32_t mkavl_avl_ctx_st_::magic

Used for sanity checks

Definition at line 64 of file mkavl.c.

4.12.2.3 mkavl_tree_handle mkavl_avl_ctx_st_::tree_h

The mkavl tree associated with the callback

Definition at line 66 of file mkavl.c.

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

- mkavl.c

4.13 mkavl_avl_tree_st_ Struct Reference

Public Attributes

- struct avl_table * tree
- mkavl_compare_fn compare_fn
- mkavl_avl_ctx_st * avl_ctx

4.13.1 Detailed Description

Maintains info on the AVL data associated with an AVL tree in the mkavl tree.

Definition at line 74 of file mkavl.c.

4.13.2 Member Data Documentation

4.13.2.1 mkavl_avl_ctx_st* mkavl_avl_tree_st_::avl_ctx

The context used for the AVL tree

Definition at line 80 of file mkavl.c.

4.13.2.2 mkavl_compare_fn mkavl_avl_tree_st_::compare_fn

The comparison function to use for the tree

Definition at line 78 of file mkavl.c.

4.13.2.3 struct avl_table* mkavl_avl_tree_st_::tree

The AVL tree pointer

Definition at line 76 of file mkavl.c.

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

- mkavl.c

4.14 mkavl_iterator_st_ Struct Reference

Public Attributes

- struct avl_traverser avl_t
- mkavl_tree_handle tree_h
- size_t key_idx

4.14.1 Detailed Description

The internal representation of the mkavl iterator.

Definition at line 127 of file mkavl.c.

4.14.2 Member Data Documentation

4.14.2.1 struct avl_traverser mkavl_iterator_st_::avl_t

The AVL traverser to use

Definition at line 129 of file mkavl.c.

4.14.2.2 size_t mkavl_iterator_st_::key_idx

The index in the mkavl tree of the AVL tree for the iteration

Definition at line 133 of file mkavl.c.

4.14.2.3 mkavl_tree_handle mkavl_iterator_st_::tree_h

The mkavl tree associaed with the iteration

Definition at line 131 of file mkavl.c.

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

- mkavl.c

4.15 mkavl_test_ctx_st_ Struct Reference

Public Attributes

- uint32_t magic
- uint32_t copy_cnt
- uint32_t item_fn_cnt
- uint32_t copy_malloc_cnt
- uint32_t copy_free_cnt

4.15.1 Detailed Description

The context storted for a tree.

Definition at line 456 of file test_mkavl.c.

4.15.2 Member Data Documentation

4.15.2.1 uint32_t mkavl_test_ctx_st_::copy_cnt

A count for how many times mkavl_test_copy_fn() was called

Definition at line 460 of file test_mkavl.c.

4.15.2.2 uint32_t mkavl_test_ctx_st_::copy_free_cnt

A count for how many time mkavl_test_copy_free() was called

Definition at line 466 of file test_mkavl.c.

4.15.2.3 uint32_t mkavl_test_ctx_st_::copy_malloc_cnt

A count for how many time mkavl_test_copy_malloc() was called

Definition at line 464 of file test_mkavl.c.

4.15.2.4 uint32_t mkavl_test_ctx_st_::item_fn_cnt

A count for how many times mkavl_test_item_fn() was called

Definition at line 462 of file test_mkavl.c.

4.15.2.5 uint32_t mkavl_test_ctx_st_::magic

A sanity check field

Definition at line 458 of file test_mkavl.c.

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

- test/test_mkavl.c

4.16 mkavl_test_input_st_ Struct Reference

Public Attributes

- uint32_t * insert_seq
- uint32_t * delete_seq
- uint32_t * sorted_seq
- uint32_t uniq_cnt
- uint32_t dup_cnt
- const test_mkavl_opts_st * opts
- mkavl_tree_handle tree_h
- mkavl_tree_handle tree_copy_h

4.16.1 Detailed Description

The input structure to pass test parameters to functions.

Definition at line 340 of file test_mkavl.c.

4.16.2 Member Data Documentation

4.16.2.1 uint32_t* mkavl_test_input_st_::delete_seq

The sequence in which items should be deleted

Definition at line 344 of file test_mkavl.c.

4.16.2.2 uint32_t mkavl_test_input_st_::dup_cnt

The count of how many duplicated items are in the sequence.

Definition at line 350 of file test_mkavl.c.

4.16.2.3 uint32_t* mkavl_test_input_st_::insert_seq

The sequence in which items should be inserted

Definition at line 342 of file test_mkavl.c.

4.16.2.4 const test_mkavl_opts_st* mkavl_test_input_st_::opts

The input options for the run

Definition at line 352 of file test_mkavl.c.

4.16.2.5 uint32_t* mkavl_test_input_st_::sorted_seq

The sequence in a sorted order

Definition at line 346 of file test_mkavl.c.

4.16.2.6 mkavl_tree_handle mkavl_test_input_st_::tree_copy_h

A deep copy of the tree (is such a copy has been done)

Definition at line 356 of file test_mkavl.c.

4.16.2.7 mkavl_tree_handle mkavl_test_input_st_::tree_h

The tree for the run

Definition at line 354 of file test_mkavl.c.

4.16.2.8 uint32_t mkavl_test_input_st_::uniq_cnt

The count of how many unique items are in the sequence.

Definition at line 348 of file test_mkavl.c.

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

- test/test_mkavl.c

4.17 mkavl_test_walk_ctx_st_ Struct Reference

Public Attributes

- uint32_t magic
- uint32_t walk_node_cnt
- uint32_t walk_stop_cnt

4.17.1 Detailed Description

The context for mkavl_walk()

Definition at line 1606 of file test_mkavl.c.

4.17.2 Member Data Documentation

4.17.2.1 uint32_t mkavl_test_walk_ctx_st_::magic

Magic value for sanity check

Definition at line 1608 of file test_mkavl.c.

4.17.2.2 uint32_t mkavl_test_walk_ctx_st_::walk_node_cnt

A count of the number of nodes walked

Definition at line 1610 of file test_mkavl.c.

4.17.2.3 uint32_t mkavl_test_walk_ctx_st_::walk_stop_cnt

Tells when the walk should be stopped

Definition at line 1612 of file test_mkavl.c.

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

- test/test_mkavl.c

4.18 mkavl_tree_st_ Struct Reference

Public Attributes

- void * context
- mkavl_allocator_wrapper_st allocator
- size_t avl_tree_count
- mkavl_avl_tree_st * avl_tree_array
- uint32_t item_count
- mkavl_copy_fn copy_fn

4.18.1 Detailed Description

The internal representation of the mkavl tree object.

Definition at line 102 of file mkavl.c.

4.18.2 Member Data Documentation

4.18.2.1 mkavl_allocator_wrapper_st mkavl_tree_st_::allocator

The memory allocator info passed in by the client

Definition at line 106 of file mkavl.c.

4.18.2.2 mkavl_avl_tree_st* mkavl_tree_st_::avl_tree_array

An array of the AVL tree info of size avl_tree_count

Definition at line 110 of file mkavl.c.

4.18.2.3 size_t mkavl_tree_st_::avl_tree_count

The number of AVL trees within this object

Definition at line 108 of file mkavl.c.

4.18.2.4 void* mkavl_tree_st_::context

The opaque context passed in by the client

Definition at line 104 of file mkavl.c.

4.18.2.5 mkavl_copy_fn mkavl_tree_st_::copy_fn

The copy function passed in my the client to apply to items when mkavl_copy is done.

Definition at line 121 of file mkavl.c.

4.18.2.6 uint32_t mkavl_tree_st_::item_count

A count of the number of data items inserted by the user. This is not the total over all AVL trees. E.g., if there are 3 AVL trees and 5 items have been inserted in the mkavl, then this value is 5, not 15.

Definition at line 116 of file mkavl.c.

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

- mkavl.c

4.19 test_mkavl_opts_st_ Struct Reference

Public Attributes

- uint32_t node_cnt
- uint32_t run_cnt
- uint32_t seed
- uint8_t verbosity
- uint32_t range_start
- uint32_t range_end

4.19.1 Detailed Description

State for the current test execution.

Definition at line 97 of file test_mkavl.c.

4.19.2 Member Data Documentation

4.19.2.1 uint32_t test_mkavl_opts_st_::node_cnt

The max number of nodes for the AVL tree

Definition at line 99 of file test_mkavl.c.

4.19.2.2 uint32_t test_mkavl_opts_st_::range_end

The ending value for the data range

Definition at line 109 of file test_mkavl.c.

4.19.2.3 uint32_t test_mkavl_opts_st_::range_start

The starting value for the data range

Definition at line 107 of file test_mkavl.c.

4.19.2.4 uint32_t test_mkavl_opts_st_::run_cnt

The number of separate runs to do

Definition at line 101 of file test_mkavl.c.

4.19.2.5 uint32_t test_mkavl_opts_st_::seed

The RNG seed for the first run

Definition at line 103 of file test_mkavl.c.

4.19.2.6 uint8_t test_mkavl_opts_st_::verbosity

The verbosity level for the test

Definition at line 105 of file test_mkavl.c.

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

- test/test_mkavl.c

Chapter 5

File Documentation

5.1 examples/employee_example.c File Reference

```
#include "examples_common.h"
#include <math.h>
```

Classes

- struct employee_example_opts_st_
- struct employee_obj_st_
- struct employee_example_input_st_
- struct employee_ctx_st_
- struct employee_walk_ctx_st_

Defines

- #define MAX_NAME_LEN 100

Typedefs

- typedef enum employee_dist_e_ employee_dist_e
- typedef struct employee_example_opts_st_ employee_example_opts_st
- typedef struct employee_obj_st_ employee_obj_st
- typedef struct employee_example_input_st_ employee_example_input_st
- typedef struct employee_ctx_st_ employee_ctx_st
- typedef struct employee_walk_ctx_st_ employee_walk_ctx_st
- typedef enum employee_example_key_e_ employee_example_key_e

Enumerations

- enum employee_dist_e_ { EMPLOYEE_DIST_E_UNIFORM, EMPLOYEE_DIST_E_ZIPF, EMPLOYEE_DIST_E_MAX }
- enum employee_example_key_e_ { EMPLOYEE_EXAMPLE_KEY_E_ID, EMPLOYEE_EXAMPLE_KEY_E_LNAME_ID, EMPLOYEE_EXAMPLE_KEY_E_MAX }

Functions

- static uint32_t zipf (double alpha, uint32_t n)
- static void print_usage (bool do_exit, int32_t exit_val)
- static void print_opts (employee_example_opts_st *opts)
- static void parse_command_line (int argc, char **argv, employee_example_opts_st *opts)
- static int32_t employee_cmp_by_id (const void *item1, const void *item2, void *context)
- static int32_t employee_cmp_by_last_name (const void *item1, const void *item2, void *context)
- static bool generate_employee (employee_example_input_st *input, employee_obj_st **obj)
- static void display_employee (employee_obj_st *obj)
- static mkavl_rc_e free_employee (void *item, void *context)
- static void lookup_employees_by_last_name (employee_example_input_st *input, const char *last_name, uint32_t max_records, bool find_all, bool do_print)
- static mkavl_rc_e last_name_walk_cb (void *item, void *tree_context, void *walk_context, bool *stop_walk)
- static void run_employee_example (employee_example_input_st *input)
- int main (int argc, char *argv[])

Variables

- static const uint32_t default_employee_cnt = 1000
- static const uint32_t default_run_cnt = 1
- static const uint8_t default_verbosity = 0
- static const employee_dist_e default_last_name_dist = EMPLOYEE_DIST_E_UNIFORM
- static const double default_zipf_alpha = 1.0
- static const char * first_names []
- static const char * last_names []
- static mkavl_compare_fn cmp_fn_array []

5.1.1 Detailed Description

Author

Matt Miller <matt@matthewjmiller.net>

5.1.2 LICENSE

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

5.1.3 DESCRIPTION

This is an example of how the mkavl library can be used. This example consists of a DB of employees where their unique ID and first and last name is stored. The first names of employees are chosen uniformly at random from a list of 100 popular names. The last names of employees are, by default, chosen uniformly at random from a list of 100 common names. There is a command-line option to use a Zipf distribution instead for the last name to give significantly more weight towards choosing the most popular names.

Running the example gives two phases: functionality and performance.

For functionality:

1. Choose ten IDs uniformly at random and lookup the employee objects.
2. Choose a last name uniformly at random and lookup up to the first ten employees with that last name. Note that this is done in *O(lg N)* time.
3. Change the last name of an employee and show that all the lookups happen as expected.

For performance:

1. Choose 30 last names uniformly at random. Lookup all the employees with each last name using the mkavl tree keyed by last name and ID (*O(lg N)*). Then, lookup all the employees with each last name in the tree by walking through all nodes (as would typically be done for a non-key field) (*O(N)*).
2. Compare the wall clock time (i.e., from gettimeofday()) for both lookup methods and the total number of nodes walked for each method.

Performance results show for 1000 employees and 100 last names, the keyed lookup shows an improvement by a factor of 10 for uniformly distributed last names. For a Zipf distribution, a majority of the runs show about a factor of 20 improvement. However, some Zipf runs only show an improvement of about 3 (presumably when some of the most popular names are randomly chosen and there are just inherently a lot of employees to lookup for that name).

```
Example of using mkavl for an employee DB

Usage:
-s <seed>
   The starting seed for the RNG (default=seeded by time()).
-n <employees>
   The number of nodes to place in the trees (default=1000).
-r <runs>
   The number of runs to do (default=1).
-v <verbosity level>
   A higher number gives more output (default=0).
-z
   Use Zipf distribution for last names (default=uniform).
-a <Zipf alpha>
   If using a Zipf distribution, the alpha value to
   parameterize the distribution (default=1.000000).
-h
   Display this help message.
```

Definition in file employee_example.c.

5.1.4 Define Documentation

5.1.4.1 #define MAX_NAME_LEN 100

Upper bound on name string lengths.

Definition at line 109 of file employee_example.c.

5.1.5 Typedef Documentation

5.1.5.1 typedef struct employee_ctx_st_ employee_ctx_st

The context associated with the employee AVLs.

5.1.5.2 typedef enum employee_dist_e_ employee_dist_e

Probability distributions used within example.

5.1.5.3 typedef struct employee_example_input_st_ employee_example_input_st

The input structure to pass test parameters to functions.

5.1.5.4 typedef enum employee_example_key_e_ employee_example_key_e

The values for the key ordering.

5.1.5.5 typedef struct employee_example_opts_st_ employee_example_opts_st

State for the current test execution.

5.1.5.6 typedef struct employee_obj_st_ employee_obj_st

The data stored for employees.

5.1.5.7 typedef struct employee_walk_ctx_st_ employee_walk_ctx_st

Context for the walk of the employee AVLs.

5.1.6 Enumeration Type Documentation

5.1.6.1 enum employee_dist_e_

Probability distributions used within example.

Enumerator:

EMPLOYEE_DIST_E_UNIFORM Uniform distribution

EMPLOYEE_DIST_E_ZIPF Zipf distribution

EMPLOYEE_DIST_E_MAX Max value for boundary checking

Definition at line 86 of file employee_example.c.

5.1.6.2 enum employee_example_key_e_

The values for the key ordering.

Enumerator:

EMPLOYEE_EXAMPLE_KEY_E_ID Ordered by ID

EMPLOYEE_EXAMPLE_KEY_E_LNAME_ID Ordered by last name + ID

EMPLOYEE_EXAMPLE_KEY_E_MAX Max value for boundary testing

Definition at line 474 of file employee_example.c.

5.1.7 Function Documentation

5.1.7.1 static void display_employee (employee_obj_st * *obj*) [static]

Display the given employee object.

Parameters

obj	The object to display.

Definition at line 549 of file employee_example.c.

5.1.7.2 static int32_t employee_cmp_by_id (const void * *item1*, const void * *item2*, void * *context*) [static]

Compare employees by ID.

Parameters

item1	Item to compare
item2	Item to compare
context	Context for the tree

Returns

Comparison result

Definition at line 412 of file employee_example.c.

5.1.7.3 static int32_t employee_cmp_by_last_name (const void * *item1*, const void * *item2*, void * *context*) [static]

Compare employees by last name and ID.

Parameters

item1	Item to compare
item2	Item to compare
context	Context for the tree

Returns

Comparison result

Definition at line 439 of file employee_example.c.

5.1.7.4 static mkavl_rc_e free_employee (void * *item*, void * *context*) [static]

Callback to free the given employee object.

Parameters

item	The pointer to the object.
context	Context for the tree.

Returns

The return code

Definition at line 567 of file employee_example.c.

5.1.7.5 static bool generate_employee (employee_example_input_st * *input*, employee_obj_st ** *obj*) [static]

Allocate and fill in the data for an employee object. The ID of the employee is a unique value for the employee. The first name is chosen from a uniform distribution of 100 names. The last name is chosen according to the given distribution of 100 names.

Parameters

input	The input parameters.
obj	A pointer to fill in with the newly allocated object.

Returns

true if the creation was successful.

Definition at line 504 of file employee_example.c.

5.1.7.6 static mkavl_rc_e last_name_walk_cb (void * *item*, void * *tree_context*, void * *walk_context*, bool * *stop_walk*) [static]

Callback for walking the tree to find all records for a given last name.

Parameters

item	The current employee object.
tree_context	The context for the tree.
walk_context	The context for the walk.
stop_walk	Setting to true will stop the walk immediately upon return.

Returns

The return code.

Definition at line 641 of file employee_example.c.

5.1.7.7 static void lookup_employees_by_last_name (employee_example_input_st * *input*, const char * *last_name*, uint32_t *max_records*, bool *find_all*, bool *do_print*) [static]

Look up a (sub)set of employees by their last name.

Parameters

input	The input parameters for the run.
last_name	The last name for which to search.
max_records	The maximum number of records to find (if find_all is false).
find_all	If true, then find all employee records for the given last name (max_records is ignored in this case).
do_print	Whether to print the info for each record.

Definition at line 590 of file employee_example.c.

5.1.7.8 int main (int *argc,* char * *argv[]*)

Main function to test objects.

Definition at line 875 of file employee_example.c.

5.1.7.9 static void parse_command_line (int *argc,* char ** *argv,* employee_example_opts_st * *opts*) [static]

Store the command line options into a local structure.

Parameters

argc	The number of options
argv	The string for the options.
opts	The local structure in which to store the parsed info.

Definition at line 321 of file employee_example.c.

5.1.7.10 static void print_opts (employee_example_opts_st * *opts*) [static]

Utility function to output the value of the options.

Parameters

opts	The options to output.

Definition at line 300 of file employee_example.c.

5.1.7.11 static void print_usage (bool *do_exit,* int32_t *exit_val*) [static]

Display the program's help screen and exit as needed.

Parameters

do_exit	Whether to exit after the output.
exit_val	If exiting the value with which to exit.

Definition at line 264 of file employee_example.c.

5.1.7.12 static void run_employee_example (employee_example_input_st * *input*) [static]

Run a single instance of an example.

Parameters

input	The input parameters for the run.

Definition at line 669 of file employee_example.c.

5.1.7.13 static uint32_t zipf (double *alpha*, uint32_t *n*) [static]

Get a random variable from a Zipf distribution within the range [1,n]. Implementation is from: http://www.cse.usf.edu/~christen/tools/genzipf.c

Parameters

alpha	The alpha paremeter for the distribution.
n	The maximum value (inclusive) for the random variable.

Returns

A value between 1 and n (inclusive).

Definition at line 215 of file employee_example.c.

5.1.8 Variable Documentation

5.1.8.1 mkavl_compare_fn cmp_fn_array[] [static]

Initial value:

```
{
    employee_cmp_by_id,
    employee_cmp_by_last_name
}
```

The comparison functions to use

Definition at line 484 of file employee_example.c.

5.1.8.2 const uint32_t default_employee_cnt = 1000 [static]

The default employee count for runs

Definition at line 96 of file employee_example.c.

5.1.8.3 const employee_dist_e default_last_name_dist = EMPLOYEE_DIST_E_UNIFORM [static]

The default last name distribution function

Definition at line 102 of file employee_example.c.

5.1.8.4 const uint32_t default_run_cnt = 1 [static]

The default number of test runs

Definition at line 98 of file employee_example.c.

5.1.8.5 const uint8_t default_verbosity = 0 [static]

The default verbosity level of messages displayed

Definition at line 100 of file employee_example.c.

5.1.8.6 const double default_zipf_alpha = 1.0 [static]

The default alpha parameter for the Zipf distribution

Definition at line 104 of file employee_example.c.

5.1.8.7 const char* first_names[] [static]

Initial value:

```
 {
    "Jacob", "Isabella", "Ethan", "Sophia", "Michael", "Emma", "Jayden",
    "Olivia", "William", "Ava", "Alexander", "Emily", "Noah", "Abigail",
    "Daniel", "Madison", "Aiden", "Chloe", "Anthony", "Mia", "Joshua",
    "Addison", "Mason", "Elizabeth", "Christopher", "Ella", "Andrew", "Natalie",
    "David", "Samantha", "Matthew", "Alexis", "Logan", "Lily", "Elijah",
    "Grace", "James", "Hailey", "Joseph", "Alyssa", "Gabriel", "Lillian",
    "Benjamin", "Hannah", "Ryan", "Avery", "Samuel", "Leah", "Jackson",
    "Nevaeh", "John", "Sofia", "Nathan", "Ashley", "Jonathan", "Anna",
    "Christian", "Brianna", "Liam", "Sarah", "Dylan", "Zoe", "Landon",
    "Victoria", "Caleb", "Gabriella", "Tyler", "Brooklyn", "Lucas", "Kaylee",
    "Evan", "Taylor", "Gavin", "Layla", "Nicholas", "Allison", "Isaac",
    "Evelyn", "Brayden", "Riley", "Luke", "Amelia", "Angel", "Khloe", "Brandon",
    "Makayla", "Jack", "Aubrey", "Isaiah", "Charlotte", "Jordan", "Savannah",
    "Owen", "Zoey", "Carter", "Bella", "Connor", "Kayla", "Justin", "Alexa"
}
```

List of first names to choose from employees

Definition at line 130 of file employee_example.c.

5.1.8.8 const char* last_names[] [static]

Initial value:

```
{
    "Smith", "Johnson", "Williams", "Jones", "Brown", "Davis", "Miller",
    "Wilson", "Moore", "Taylor", "Anderson", "Thomas", "Jackson", "White",
    "Harris", "Martin", "Thompson", "Garcia", "Martinez", "Robinson", "Clark",
    "Rodriguez", "Lewis", "Lee", "Walker", "Hall", "Allen", "Young",
    "Hernandez", "King", "Wright", "Lopez", "Hill", "Scott", "Green", "Adams",
    "Baker", "Gonzalez", "Nelson", "Carter", "Mitchell", "Perez", "Roberts",
    "Turner", "Phillips", "Campbell", "Parker", "Evans", "Edwards", "Collins",
    "Stewart", "Sanchez", "Morris", "Rogers", "Reed", "Cook", "Morgan", "Bell",
    "Murphy", "Bailey", "Rivera", "Cooper", "Richardson", "Cox", "Howard",
    "Ward", "Torres", "Peterson", "Gray", "Ramirez", "James", "Watson",
    "Brooks", "Kelly", "Sanders", "Price", "Bennett", "Wood", "Barnes", "Ross",
    "Henderson", "Coleman", "Jenkins", "Perry", "Powell", "Long", "Patterson",
    "Hughes", "Flores", "Washington", "Butler", "Simmons", "Foster", "Gonzales",
    "Bryant", "Alexander", "Russell", "Griffin", "Diaz", "Hayes"
}
```

List of last names to choose from employees

Definition at line 148 of file employee_example.c.

5.2 examples/examples_common.h File Reference

```
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <unistd.h>
#include <assert.h>
#include <stdio.h>
#include <time.h>
#include <errno.h>
#include <sys/time.h>
#include "../mkavl.h"
```

Defines

- #define NELEMS(x) (sizeof(x) / sizeof(x[0]))
- #define CT_ASSERT(e) extern char (*CT_ASSERT(void)) [sizeof(char[1 - 2*!(e)])]
- #define EXAMPLES_RUNAWAY_SANITY 100000

Functions

- static void assert_abort (bool condition)
- static double timeval_to_seconds (struct timeval *tv)

- static size_t my_strlcpy (char *dst, const char *src, size_t siz)

5.2.1 Detailed Description

Author

Matt Miller <matt@matthewjmiller.net>

5.2.2 LICENSE

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

5.2.3 DESCRIPTION

These are common functionalities shared by the examples.

Definition in file examples_common.h.

5.2.4 Define Documentation

5.2.4.1 #define CT_ASSERT(*e*) extern char (*CT_ASSERT(void)) [sizeof(char[1 - 2*!(e)])]

Compile time assert macro from: http://www.pixelbeat.org/programming/gcc/static_assert.html

Definition at line 51 of file examples_common.h.

5.2.4.2 #define EXAMPLES_RUNAWAY_SANITY 100000

Sanity check for infinite loops.

Definition at line 57 of file examples_common.h.

5.2.4.3 #define NELEMS(*x*) (sizeof(x) / sizeof(x[0]))

Determine the number of elements in an array.

Definition at line 43 of file examples_common.h.

5.2.5 Function Documentation

5.2.5.1 static void assert_abort (bool *condition*) [inline, static]

Assert utility to crash (via abort()) if the condition is not met regardless of whether NDEBUG is defined.

Parameters

condition	The condition for which a crash will happen if false.

Definition at line 66 of file examples_common.h.

5.2.5.2 static size_t my_strlcpy (char * *dst*, const char * *src*, size_t *siz*) [inline, static]

Sigh, yes for reasons divorced from reality, you just have to keep implementing this. Copied from the BSD source. See Wikipedia for more documentation.

Parameters

dst	The destination string.
src	The source string.
siz	The size of the destination buffer.

Returns

The length of the source string.

Definition at line 101 of file examples_common.h.

5.2.5.3 static double timeval_to_seconds (struct timeval * *tv*) [inline, static]

Utility to convert time objects to seconds.

Parameters

tv	The time object.

Returns

The value in seconds.

Definition at line 80 of file examples_common.h.

5.3 examples/malloc_example.c File Reference

```
#include "examples_common.h"
```

Classes

- struct malloc_example_opts_st_
- struct memblock_obj_st_
- struct malloc_example_input_st_
- struct memblock_ctx_st_

Typedefs

- typedef enum malloc_pattern_e_ malloc_pattern_e
- typedef struct malloc_example_opts_st_ malloc_example_opts_st
- typedef struct memblock_obj_st_ memblock_obj_st
- typedef struct malloc_example_input_st_ malloc_example_input_st
- typedef struct memblock_ctx_st_ memblock_ctx_st
- typedef enum malloc_example_key_e_ malloc_example_key_e

Enumerations

- enum malloc_pattern_e_ { MALLOC_PATTERN_E_LINEAR, MALLOC_PATTERN_-E_UNIFORM, MALLOC_PATTERN_E_MAX }
- enum malloc_example_key_e_ { MALLOC_EXAMPLE_KEY_E_ADDR, MALLOC_-EXAMPLE_KEY_E_SIZE, MALLOC_EXAMPLE_KEY_E_MAX }

Functions

- static void print_usage (bool do_exit, int32_t exit_val)
- static void print_opts (malloc_example_opts_st ∗opts)
- static void parse_command_line (int argc, char ∗∗argv, malloc_example_opts_st ∗opts)
- static int32_t memblock_cmp_by_addr (const void ∗item1, const void ∗item2, void ∗context)
- static int32_t memblock_cmp_by_size (const void ∗item1, const void ∗item2, void ∗context)
- static mkavl_rc_e free_memblock (void ∗item, void ∗context)
- static void display_memory (mkavl_tree_handle tree_h, void ∗start_addr, size_t bytes)
- static bool generate_memblock (memblock_obj_st ∗∗obj, void ∗start_addr, size_t byte_cnt)
- static void ∗ my_malloc (mkavl_tree_handle tree_h, size_t size)
- static void my_free (mkavl_tree_handle tree_h, void ∗ptr)
- static void run_malloc_example (malloc_example_input_st ∗input)
- int main (int argc, char ∗argv[])

Variables

- static const size_t malloc_sizes [] = { 4, 8, 512, 4096 }
- static const uint32_t default_malloc_cnt = 100
- static uint32_t default_memory_size
- static const uint32_t default_run_cnt = 1
- static const uint8_t default_verbosity = 0
- static void * base_addr = (void *) 0x1234ABCD
- static size_t max_memory_size
- static mkavl_compare_fn cmp_fn_array []

5.3.1 Detailed Description

Author

Matt Miller <matt@matthewjmiller.net>

5.3.2 LICENSE

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

5.3.3 DESCRIPTION

This is a basic example of how the mkavl library can be used for memory management. The free and allocated memory blocks are maintained in a single mvavl DB. The DB is indexed by the starting address of the memory block as one key and the other key consists of the allocation status (i.e., free or allocated), block size, and starting address.

On a malloc() call, we look up the free block with the size greater than or equal to the requested size. This is a O(lg N) best-fit algorithm. On a free() call, we change the state of the freed block from allocated to free. We then check whether the adjacent memory blocks are also free and, if so, consolidate the blocks into one.

The example run will:

1. Allocate 100 pointers
2. Free up to half of them.
3. Re-allocate the ones just freed.
4. Free all the pointers.

At each step, we print out a graphical display of the curent memory state. The step where up to half are freed is done by chosing points uniformly at random by default. A command line option allows you to instead free the first half of the pointers deterministically.

Of course, this is just an example so we use malloc to generate the AVL items placed in the tree. In reality, a more complicated scheme would be implemented to grab memory for the purpose so malloc isn't being called to implement malloc.

```
Example of using mkavl for an memory allocation

Usage:
-s <seed>
   The starting seed for the RNG (default=seeded by time()).
-b <memory size in bytes>
   The number of bytes in memory (default=409600).
-n <number of allocations>
   The max number of allocations at any one time (default=100).
-r <runs>
   The number of runs to do (default=1).
-l
   Free/re-allocate linearly (default=uniform distribution).
-v <verbosity level>
   A higher number gives more output (default=0).
-h
   Display this help message.
```

Definition in file malloc_example.c.

5.3.4 Typedef Documentation

5.3.4.1 typedef struct malloc_example_input_st_ malloc_example_input_st

The input structure to pass test parameters to functions.

5.3.4.2 typedef enum malloc_example_key_e_ malloc_example_key_e

The values for the key ordering.

5.3.4.3 typedef struct malloc_example_opts_st_ malloc_example_opts_st

State for the current test execution.

5.3.4.4 typedef enum malloc_pattern_e_ malloc_pattern_e

Patterns for how memory gets freed and re-allocated.

5.3.4.5 typedef struct memblock_ctx_st_ memblock_ctx_st

The context associated with the memblock AVLs.

5.3.4.6 typedef struct memblock_obj_st_ memblock_obj_st

The data for a free/allocated memory block.

5.3.5 Enumeration Type Documentation

5.3.5.1 enum malloc_example_key_e_

The values for the key ordering.

Enumerator:

MALLOC_EXAMPLE_KEY_E_ADDR Ordered by address

MALLOC_EXAMPLE_KEY_E_SIZE Ordered by allocation status + size + address

MALLOC_EXAMPLE_KEY_E_MAX Max value for boundary testing

Definition at line 372 of file malloc_example.c.

5.3.5.2 enum malloc_pattern_e_

Patterns for how memory gets freed and re-allocated.

Enumerator:

MALLOC_PATTERN_E_LINEAR Free and re-allocate the first N memory locations

MALLOC_PATTERN_E_UNIFORM Free and re-allocated the memory in locations chosen from a uniform distribution.

MALLOC_PATTERN_E_MAX Max value for boundary checking

Definition at line 93 of file malloc_example.c.

5.3.6 Function Documentation

5.3.6.1 static void display_memory (mkavl_tree_handle *tree_h*, void * *start_addr*, size_t *bytes*) [static]

Display memory in the given range.

Parameters

tree_h	The memory block trees.
start_addr	The start address of the range.
bytes	The number of bytes to display.

Definition at line 418 of file malloc_example.c.

5.3.6.2 static mkavl_rc_e free_memblock (void * *item*, void * *context*) [static]

Callback to free the given memory block object.

Parameters

item	The pointer to the object.
context	Context for the tree.

Returns

The return code

Definition at line 399 of file malloc_example.c.

5.3.6.3 static bool generate_memblock (memblock_obj_st ** *obj*, void * *start_addr*, size_t *byte_cnt*) [static]

Allocate and fill in the data for a memory block object. By default, the object is set to not allocated.

Obviously in a real implementation, malloc() wouldn't be used to implement malloc. You'd need to slice up the memory available yourself.

Parameters

obj	A pointer to fill in with the newly allocated object.
start_addr	The start address for the block.
byte_cnt	The byte count for the block.

Returns

true if the creation was successful.

Definition at line 471 of file malloc_example.c.

5.3.6.4 int main (int *argc*, char * *argv[]*)

Main function to test objects.

Definition at line 739 of file malloc_example.c.

5.3.6.5 static int32_t memblock_cmp_by_addr (const void * *item1*, const void * *item2*, void * *context*) [static]

Compare memory blocks by address.

Parameters

item1	Item to compare
item2	Item to compare
context	Context for the tree

Returns

Comparison result

Definition at line 309 of file malloc_example.c.

5.3.6.6 static int32_t memblock_cmp_by_size (const void * *item1*, const void * *item2*, void * *context*) [static]

Compare memory blocks by allocated status, size, and address.

Parameters

item1	Item to compare
item2	Item to compare
context	Context for the tree

Returns

Comparison result

Definition at line 336 of file malloc_example.c.

5.3.6.7 static void my_free (mkavl_tree_handle *tree_h*, void * *ptr*) [static]

Mark the memory as unallocated and merge with adjacent unallocated blocks.

Parameters

tree_h	The tree of memory blocks.
ptr	The pointer to free.

Definition at line 571 of file malloc_example.c.

5.3.6.8 static void* my_malloc (mkavl_tree_handle *tree_h*, size_t *size*) [static]

This is a best-fit version that will find the first unallocated memory block large enough to hold the request.

Parameters

tree_h	The tree of memory blocks.
size	The size of memory to allocate.

Returns

A pointer to the memory of NULL if the allocation failed.

Definition at line 504 of file malloc_example.c.

5.3.6.9 static void parse_command_line (int *argc,* char ** *argv,* malloc_example_opts_st * *opts*) [static]

Store the command line options into a local structure.

Parameters

argc	The number of options
argv	The string for the options.
opts	The local structure in which to store the parsed info.

Definition at line 219 of file malloc_example.c.

5.3.6.10 static void print_opts (malloc_example_opts_st * *opts*) [static]

Utility function to output the value of the options.

Parameters

opts	The options to output.

Definition at line 198 of file malloc_example.c.

5.3.6.11 static void print_usage (bool *do_exit,* int32_t *exit_val*) [static]

Display the program's help screen and exit as needed.

Parameters

do_exit	Whether to exit after the output.
exit_val	If exiting the value with which to exit.

Definition at line 163 of file malloc_example.c.

5.3.6.12 static void run_malloc_example (malloc_example_input_st * *input*) [static]

Run a single instance of an example.

Parameters

input	The input parameters for the run.

Definition at line 646 of file malloc_example.c.

5.3.7 Variable Documentation

5.3.7.1 void* base_addr = (void *) 0x1234ABCD [static]

The address to use as the base of the memory

Definition at line 86 of file malloc_example.c.

5.3.7.2 mkavl_compare_fn cmp_fn_array[] [static]

Initial value:

```
 {
    memblock_cmp_by_addr,
    memblock_cmp_by_size
}
```

The comparison functions to use

Definition at line 382 of file malloc_example.c.

5.3.7.3 const uint32_t default_malloc_cnt = 100 [static]

The default number of items to allocated at any one time

Definition at line 77 of file malloc_example.c.

5.3.7.4 uint32_t default_memory_size [static]

The default memory size

Definition at line 79 of file malloc_example.c.

5.3.7.5 const uint32_t default_run_cnt = 1 [static]

The default number of test runs

Definition at line 81 of file malloc_example.c.

5.3.7.6 const uint8_t default_verbosity = 0 [static]

The default verbosity level of messages displayed

Definition at line 83 of file malloc_example.c.

5.3.7.7 const size_t malloc_sizes[] = { 4, 8, 512, 4096 } [static]

List of sizes for memory allocations

Definition at line 74 of file malloc_example.c.

5.3.7.8 size_t max_memory_size [static]

Maximum size of the memory

Definition at line 88 of file malloc_example.c.

5.4 mkavl.c File Reference

```
#include "mkavl.h"
#include "libavl/avl.h"
#include <stdio.h>
```

Classes

- struct mkavl_avl_ctx_st_
- struct mkavl_avl_tree_st_
- struct mkavl_allocator_wrapper_st_
- struct mkavl_tree_st_
- struct mkavl_iterator_st_

Defines

- #define CT_ASSERT(e) extern char (∗CT_ASSERT(void)) [sizeof(char[1 - 2∗!(e)])]
- #define NELEMS(x) (sizeof(x) / sizeof(x[0]))
- #define MKAVL_RUNAWAY_SANITY 100000
- #define MKAVL_CTX_MAGIC 0xCAFEBABE
- #define MKAVL_CTX_STALE 0xDEADBEEF

Typedefs

- typedef struct mkavl_avl_ctx_st_ mkavl_avl_ctx_st
- typedef struct mkavl_avl_tree_st_ mkavl_avl_tree_st
- typedef struct mkavl_allocator_wrapper_st_ mkavl_allocator_wrapper_st
- typedef struct mkavl_tree_st_ mkavl_tree_st
- typedef struct mkavl_iterator_st_ mkavl_iterator_st

Functions

- static void mkavl_assert_abort (bool condition)
- bool mkavl_rc_e_is_notok (mkavl_rc_e rc)
- bool mkavl_rc_e_is_ok (mkavl_rc_e rc)
- bool mkavl_rc_e_is_valid (mkavl_rc_e rc)
- const char * mkavl_rc_e_get_string (mkavl_rc_e rc)
- bool mkavl_find_type_e_is_valid (mkavl_find_type_e type)
- const char * mkavl_find_type_e_get_string (mkavl_find_type_e type)
- static bool mkavl_avl_ctx_is_valid (mkavl_avl_ctx_st *avl_ctx)
- static bool mkavl_allocator_wrapper_is_valid (mkavl_allocator_wrapper_st *allocator)
- static void * mkavl_malloc_wrapper (struct libavl_allocator *allocator, size_t size)
- static void mkavl_free_wrapper (struct libavl_allocator *allocator, void *libavl_block)
- static void * mkavl_default_malloc_fn (size_t size, void *context)
- static void mkavl_default_free_fn (void *ptr, void *context)
- static int mkavl_compare_wrapper (const void *avl_a, const void *avl_b, void *avl_param)
- static void * mkavl_copy_wrapper (void *avl_item, void *avl_param)
- static mkavl_rc_e mkavl_delete_tree (mkavl_tree_handle *tree_h)
- static bool mkavl_tree_is_valid (mkavl_tree_handle tree_h)
- static bool mkavl_iterator_is_valid (mkavl_iterator_handle iterator_h)
- mkavl_rc_e mkavl_new (mkavl_tree_handle *tree_h, mkavl_compare_fn *compare_fn_array, size_t compare_fn_array_count, void *context, mkavl_allocator_st *allocator)
- void * mkavl_get_tree_context (mkavl_tree_handle tree_h)
- mkavl_rc_e mkavl_delete (mkavl_tree_handle *tree_h, mkavl_item_fn item_fn, mkavl_delete_context_fn delete_context_fn)
- mkavl_rc_e mkavl_copy (mkavl_tree_handle source_tree_h, mkavl_tree_handle *new_tree_h, mkavl_copy_fn copy_fn, mkavl_item_fn item_fn, bool use_source_context, void *new_context, mkavl_delete_context_fn delete_context_fn, mkavl_allocator_st *allocator)
- mkavl_rc_e mkavl_add (mkavl_tree_handle tree_h, void *item_to_add, void **existing_item)
- static mkavl_rc_e mkavl_avl_end_item (const struct avl_node *node, void **found_item, uint8_t side)
- static mkavl_rc_e mkavl_avl_first (const struct avl_node *node, void **found_item)
- static mkavl_rc_e mkavl_avl_last (const struct avl_node *node, void **found_item)
- static mkavl_rc_e mkavl_find_avl_lt (const struct avl_table *avl_tree, const struct avl_node *node, bool find_equal_to, const void *lookup_item, void **found_item)
- static mkavl_rc_e mkavl_find_avl_gt (const struct avl_table *avl_tree, const struct avl_node *node, bool find_equal_to, const void *lookup_item, void **found_item)
- mkavl_rc_e mkavl_find (mkavl_tree_handle tree_h, mkavl_find_type_e type, size_t key_idx, const void *lookup_item, void **found_item)
- mkavl_rc_e mkavl_remove (mkavl_tree_handle tree_h, const void *item_to_remove, void **found_item)

- mkavl_rc_e mkavl_add_key_idx (mkavl_tree_handle tree_h, size_t key_idx, void *item_to_add, void **existing_item)
- mkavl_rc_e mkavl_remove_key_idx (mkavl_tree_handle tree_h, size_t key_idx, const void *item_to_remove, void **found_item)
- uint32_t mkavl_count (mkavl_tree_handle tree_h)
- mkavl_rc_e mkavl_walk (mkavl_tree_handle tree_h, mkavl_walk_cb_fn cb_fn, void *walk_context)
- mkavl_rc_e mkavl_iter_new (mkavl_iterator_handle *iterator_h, mkavl_tree_handle tree_h, size_t key_idx)
- mkavl_rc_e mkavl_iter_delete (mkavl_iterator_handle *iterator_h)
- mkavl_rc_e mkavl_iter_first (mkavl_iterator_handle iterator_h, void **item)
- mkavl_rc_e mkavl_iter_last (mkavl_iterator_handle iterator_h, void **item)
- mkavl_rc_e mkavl_iter_find (mkavl_iterator_handle iterator_h, void *lookup_item, void **found_item)
- mkavl_rc_e mkavl_iter_next (mkavl_iterator_handle iterator_h, void **item)
- mkavl_rc_e mkavl_iter_prev (mkavl_iterator_handle iterator_h, void **item)
- mkavl_rc_e mkavl_iter_cur (mkavl_iterator_handle iterator_h, void **item)

Variables

- static const char *const mkavl_rc_e_string []
- static const char *const mkavl_find_type_e_string []
- static mkavl_allocator_st mkavl_allocator_default
- static struct libavl_allocator mkavl_allocator_wrapper

5.4.1 Detailed Description

Author

Matt Miller <matt@matthewjmiller.net>

5.4.2 LICENSE

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

5.4.3 DESCRIPTION

This is the implementation for the multi-key AVL interface.

Definition in file mkavl.c.

5.4.4 Define Documentation

5.4.4.1 #define CT_ASSERT(e) extern char (*CT_ASSERT(void)) [sizeof(char[1 - 2*!(e)])]

Compile time assert macro from: http://www.pixelbeat.org/programming/gcc/static_assert.html

Definition at line 34 of file mkavl.c.

5.4.4.2 #define MKAVL_CTX_MAGIC 0xCAFEBABE

Magic number indicating a pointer is valid for sanity checks.

Definition at line 52 of file mkavl.c.

5.4.4.3 #define MKAVL_CTX_STALE 0xDEADBEEF

Magic number indicating a pointer is stale for sanity checks.

Definition at line 57 of file mkavl.c.

5.4.4.4 #define MKAVL_RUNAWAY_SANITY 100000

Sanity value to make sure we don't get stuck in an infinite loop.

Definition at line 47 of file mkavl.c.

5.4.4.5 #define NELEMS(x) (sizeof(x) / sizeof(x[0]))

Determine the number of elements in an array.

Definition at line 41 of file mkavl.c.

5.4.5 Typedef Documentation

5.4.5.1 typedef struct mkavl_allocator_wrapper_st_ mkavl_allocator_wrapper_st

A wrapper structure to map the mkavl_allocator to the avl_allocator.

5.4.5.2 typedef struct mkavl_avl_ctx_st_ mkavl_avl_ctx_st

The internal context data for AVL callbacks.

5.4.5.3 typedef struct mkavl_avl_tree_st_ mkavl_avl_tree_st

Maintains info on the AVL data associated with an AVL tree in the mkavl tree.

5.4.5.4 typedef struct mkavl_iterator_st_ mkavl_iterator_st

The internal representation of the mkavl iterator.

5.4.5.5 typedef struct mkavl_tree_st_ mkavl_tree_st

The internal representation of the mkavl tree object.

5.4.6 Function Documentation

5.4.6.1 mkavl_rc_e mkavl_add (mkavl_tree_handle *tree_h*, void * *item_to_add*, void ** *existing_item*)

Add an item to each AVL tree in the mkavl.

Parameters

tree_h	The mkavl tree.
item_to_add	A pointer to the data to add.
existing_item	If an existing item is found, it is returned. Otherwise, NULL if the item was not found.

Returns

The return code

Definition at line 1004 of file mkavl.c.

5.4.6.2 mkavl_rc_e mkavl_add_key_idx (mkavl_tree_handle *tree_h*, size_t *key_idx*, void * *item_to_add*, void ** *existing_item*)

Add an item only to a specific AVL tree within the mkavl tree. Note that this must be used very carefully in conjunction with mkavl_remove_key_idx to make sure the mkavl does not become out of sync. In steady state, the mkavl should always have an equal number of data items in each AVL tree. However, if a field changes in an item that affects some keys but not other, these APIs may be used to remove the item with the old values from the affected AVLs, update the values, and then re-add to each AVL tree from which the old item was removed.

See also

mkavl_remove_key_idx

Parameters

tree_h	The mkavl tree.
key_idx	The index of the AVL tree to which the item is added.
item_to_add	The item being added.
existing_item	If an existing item is found, it is returned.

Returns

The return code

Definition at line 1422 of file mkavl.c.

5.4.6.3 static bool mkavl_allocator_wrapper_is_valid (mkavl_allocator_wrapper_st * *allocator*) [static]

Sanity check for mkavl_allocator_wrapper_st objects.

Parameters

allocator	The object to check. If NULL, false is returned.

Returns

true if the object is valid.

Definition at line 314 of file mkavl.c.

5.4.6.4 static void mkavl_assert_abort (bool *condition*) [inline, static]

Assert utility to crash (via abort()) if the condition is not met regardless of whether NDEBUG is defined. E.g., if we hit an error condition where the data structures are irreversibly out of sync.

Parameters

condition	The condition for which a crash will happen if false.

Definition at line 144 of file mkavl.c.

5.4.6.5 static bool mkavl_avl_ctx_is_valid (mkavl_avl_ctx_st * *avl_ctx*) [static]

Sanity check for mkavl_avl_ctx_st objects.

Parameters

avl_ctx	The object to check. If NULL, false is returned.

Returns

true if the object is valid.

Definition at line 284 of file mkavl.c.

5.4.6.6 static mkavl_rc_e mkavl_avl_end_item (const struct avl_node * *node*, void ** *found_item*, uint8_t *side*) [inline, static]

Get the item at the end of the AVL tree. That is, either the first or last item in the tree rooted at the node depending on the side input.

Parameters

node	The root of the subtree to search.
found_item	The item found.
side	0 to find the first (smallest) item or 1 to find the last (largest) item

Returns

The return code

Definition at line 1063 of file mkavl.c.

5.4.6.7 static mkavl_rc_e mkavl_avl_first (const struct avl_node * *node*, void ** *found_item*) [static]

Get the first (smallest) item in the tree rooted at the node.

Parameters

node	The root of the subtree.
found_item	The item that was found.

Returns

The return code

Definition at line 1098 of file mkavl.c.

5.4.6.8 static mkavl_rc_e mkavl_avl_last (const struct avl_node * *node*, void ** *found_item*) [static]

Get the last (largest) item in the tree rooted at the node.

Parameters

node	The root of the subtree.
found_item	The item that was found.

Returns

The return code

Definition at line 1111 of file mkavl.c.

5.4.6.9 static int mkavl_compare_wrapper (const void * *avl_a,* const void * *avl_b,* void * *avl_param*) [static]

A wrapper to map the AVL callback to the client callback for the mkavl tree.

Parameters

avl_a	The first item in the comparison
avl_b	The second item in the comparison
avl_param	The context associated with the callback

Returns

0 if equal, -1 if avl_a $<$ avl_b, and 1 if avl_a $>$ avl_b

Definition at line 428 of file mkavl.c.

5.4.6.10 mkavl_rc_e mkavl_copy (mkavl_tree_handle *source_tree_h,* mkavl_tree_handle * *new_tree_h,* mkavl_copy_fn *copy_fn,* mkavl_item_fn *item_fn,* bool *use_source_context,* void * *new_context,* mkavl_delete_context_fn *delete_context_fn,* mkavl_allocator_st * *allocator*)

Deep copy a mkavl tree into a new tree.

Parameters

source_tree_h	The tree from which to copy.
new_tree_h	A pointer to the new tree to which the copy will be done.
copy_fn	A function that is applied to each item in the source tree before it is copied to the new tree. This is applied once per item regardless of how many AVL trees there are. E.g., this may allocate a deep copy of the data. If NULL, then a shallow copy of the data is copied to the new tree. Note that this function is called with the source tree's context value.
item_fn	If there is an error copying the new tree, this function is applied to all items in the new tree as it is destroyed. E.g., this may be a function to free the data.
use_source_context	If true, the client context from source_tree_h is used for the new tree. If false, new_context is used for the new tree.
new_context	The opaque client context to use for the new tree if user_source_context if false.
delete_context_fn	Upon an error, this function will be applied to the new_context if use_source_context is false. If use_source_context is true, this is a no-op in the case of an error (will not call on the source tree's context). If NULL, no function is applied. If given, the function will be called even if the client context is NULL.
allocator	The memory allocation functions to use for the new tree.

Returns

The return code

Definition at line 862 of file mkavl.c.

5.4.6.11 static void* mkavl_copy_wrapper (void * *avl_item*, void * *avl_param*) [static]

A wrapper to map the AVL callback to the client callback for the mkavl tree.

Parameters

avl_item	The AVL item to copy
avl_param	The context associated with the callback

Returns

The new, copied item

Definition at line 450 of file mkavl.c.

5.4.6.12 uint32_t mkavl_count (mkavl_tree_handle *tree_h*)

Get a count of the total number of items within the tree. Note that this is the steady state account, i.e., broadly, the number of calls to mkavl_add that succeeded minus the number of calls to mkavl_remove that succeeded. If this is called in between calls to mkavl_add_key_idx and mkavl_remove_key_idx, then it is possible for individual AVL trees to have a different count.

See also

mkavl_add
mkavl_remove
mkavl_add_key_idx
mkavl_remove_key_idx

Definition at line 1502 of file mkavl.c.

5.4.6.13 static void mkavl_default_free_fn (void * *ptr*, void * *context*) [static]

The default free function.

Parameters

ptr	The memory to free.
context	The tree context.

Definition at line 398 of file mkavl.c.

5.4.6.14 static void* mkavl_default_malloc_fn (size_t *size*, void * *context*) [static]

The default malloc function.

Parameters

size	Size of memory to allocate.
context	The tree context.

Returns

A pointer to the memory or NULL if allocation was not possible.

Definition at line 386 of file mkavl.c.

5.4.6.15 mkavl_rc_e mkavl_delete (mkavl_tree_handle * *tree_h*, mkavl_item_fn *item_fn*, mkavl_delete_context_fn *delete_context_fn*)

The destroys the tree that was allocated by mkavl_new. Note that this doesn't actually free the data of the items as that is left to the client. The item_fn parameter can be used for this purpose. Upon return, the tree_h memory is set to NULL. This is O(N lg N) for N items.

See also

mkavl_new
mkavl_delete_tree

Parameters

tree_h	A pointer the the tree to free.
item_fn	This function is applied to each item after it has been removed from all the AVL trees. This is only called once per item regardless of how many AVL trees there are. E.g., this could be the function to free the memory for the data. If NULL, no function is applied.
delete_context_fn	This function is applied to the tree's opaque client context that was given via mkavl_new(). E.g., this could free the client context if it was a heap memory pointer. If NULL, no function is applied. If given, the function will be called even if the client context is NULL. This function is called after all the items have been deleted and item functions called.

Returns

The return code

Definition at line 750 of file mkavl.c.

5.4.6.16 static mkavl_rc_e mkavl_delete_tree (mkavl_tree_handle * *tree_h*) [static]

This will free all memory associated with the tree and set the pointer to NULL.

Parameters

tree_h	Pointer to the tree to delete

Returns

The return code

Definition at line 472 of file mkavl.c.

5.4.6.17 mkavl_rc_e mkavl_find (mkavl_tree_handle *tree_h*, mkavl_find_type_e *type*, size_t *key_idx*, const void * *lookup_item*, void ** *found_item*)

Find an item in the tree.

Parameters

tree_h	The tree to search
type	The type of lookup to do.
key_idx	The AVL tree being searched.
lookup_item	The item to use as the lookup target.
found_item	The item found for the lookup.

Returns

The return code

Definition at line 1275 of file mkavl.c.

5.4.6.18 static mkavl_rc_e mkavl_find_avl_gt (const struct avl_table * *avl_tree*, const struct avl_node * *node*, bool *find_equal_to*, const void * *lookup_item*, void ** *found_item*) [static]

Find the item greater than (or equal to) the given lookup_item. Note that the lookup_item does not necessarily have to exist in the tree for an item to be found. This is a recursive funciton.

Parameters

avl_tree	The AVL tree to search.
node	The current node to compare.
find_equal_to	Indicates whether to return an equal item if found or only strictly greater than.
lookup_item	The item to use as the lookup target.
found_item	The item found.

Returns

The return code

Definition at line 1204 of file mkavl.c.

5.4.6.19 static mkavl_rc_e mkavl_find_avl_lt (const struct avl_table * *avl_tree,* const struct avl_node * *node,* bool *find_equal_to,* const void * *lookup_item,* void ** *found_item*) [static]

Find the item less than (or equal to) the given lookup_item. Note that the lookup_item does not necessarily have to exist in the tree for an item to be found. This is a recursive funciton.

Parameters

avl_tree	The AVL tree to search.
node	The current node to compare.
find_equal_to	Indicates whether to return an equal item if found or only strictly less than.
lookup_item	The item to use as the lookup target.
found_item	The item found.

Returns

The return code

Definition at line 1130 of file mkavl.c.

5.4.6.20 const char* mkavl_find_type_e_get_string (mkavl_find_type_e *type*)

Get a string representation of the find type.

Parameters

type	The find type

Returns

A string representation of the return code or "__Invalid__" if an invalid return code is input.

Definition at line 266 of file mkavl.c.

5.4.6.21 bool mkavl_find_type_e_is_valid (mkavl_find_type_e *type*)

Indicates whether the return code is valid.

Parameters

type	The type to check

Returns

true if the return code is valid.

Definition at line 252 of file mkavl.c.

5.4.6.22 static void mkavl_free_wrapper (struct libavl_allocator * *allocator,* void * *libavl_block*) [static]

A wrapper to map the AVL callback to the client callback for the mkavl tree.

Parameters

allocator	The memory allocator associated with the callback
libavl_block	The memory to free

Definition at line 367 of file mkavl.c.

5.4.6.23 void* mkavl_get_tree_context (mkavl_tree_handle *tree_h*)

Get the context pointer for the tree.

See also

mkavl_new

Parameters

tree_h	The tree whose context to get. This must be a valid, non-NULL tree pointer or else a crash will occur.

Returns

The tree's context (possibly NULL if that was passed to mkavl_new()).

Definition at line 721 of file mkavl.c.

5.4.6.24 mkavl_rc_e mkavl_iter_cur (mkavl_iterator_handle *iterator_h,* void ** *item*)

Get the current item in the iteration.

See also

mkavl_iter_new

Parameters

iterator_h	The iterator to use.
item	The item found.

Returns

The return code

Definition at line 1780 of file mkavl.c.

5.4.6.25 mkavl_rc_e mkavl_iter_delete (mkavl_iterator_handle * *iterator_h*)

Destroy the iterator.

See also

mkavl_iter_new

Parameters

iterator_h	The iterator to free. Upon return, this will be set to NULL.

Returns

The return code

Definition at line 1617 of file mkavl.c.

5.4.6.26 mkavl_rc_e mkavl_iter_find (mkavl_iterator_handle *iterator_h*, void * *lookup_item*, void ** *found_item*)

Find an item and update the iterator to that node.

See also

mkavl_iter_new

Parameters

iterator_h	The iterator to use.
lookup_item	The item to find.
found_item	The item found.

Returns

The return code

Definition at line 1701 of file mkavl.c.

5.4.6.27 mkavl_rc_e mkavl_iter_first (mkavl_iterator_handle *iterator_h*, void ** *item*)

Get the first item in the iteration.

See also

mkavl_iter_new

Parameters

iterator_h	The iterator to use.
item	The item found.

Returns

The return code

Definition at line 1648 of file mkavl.c.

5.4.6.28 mkavl_rc_e mkavl_iter_last (mkavl_iterator_handle *iterator_h*, void ** *item*)

Get the last item in the iteration.

See also

mkavl_iter_new

Parameters

iterator_h	The iterator to use.
item	The item found.

Returns

The return code

Definition at line 1674 of file mkavl.c.

5.4.6.29 mkavl_rc_e mkavl_iter_new (mkavl_iterator_handle * *iterator_h*, mkavl_tree_handle *tree_h*, size_t *key_idx*)

Create a new iterator to use.

See also

mkavl_iter_delete

Parameters

iterator_h	The pointer to fill in with the new iterator.
tree_h	The tree on which to iterate.
key_idx	The index of the AVL tree on which to iterate.

Returns

The return code

Definition at line 1565 of file mkavl.c.

5.4.6.30 mkavl_rc_e mkavl_iter_next (mkavl_iterator_handle *iterator_h*, void ** *item*)

Get the next item in the iteration and update the iterator to that node.

See also

mkavl_iter_new

Parameters

iterator_h	The iterator to use.
item	The item found.

Returns

The return code

Definition at line 1730 of file mkavl.c.

5.4.6.31 mkavl_rc_e mkavl_iter_prev (mkavl_iterator_handle *iterator_h*, void ** *item*)

Get the previous item in the iteration and update the iterator to that node.

See also

mkavl_iter_new

Parameters

iterator_h	The iterator to use.
item	The item found.

Returns

The return code

Definition at line 1755 of file mkavl.c.

5.4.6.32 static bool mkavl_iterator_is_valid (mkavl_iterator_handle *iterator_h*) [static]

Sanity check for mkavl_iterator_handle objects.

Parameters

iterator_h	The object to check. If NULL, false is returned.

Returns

true if the object is valid.

Definition at line 570 of file mkavl.c.

5.4.6.33 static void* mkavl_malloc_wrapper (struct libavl_allocator * *allocator*, size_t *size*) [static]

A wrapper to map the AVL callback to the client callback for the mkavl tree.

Parameters

allocator	The memory allocator associated with the callback
size	The size to allocate

Returns

A pointer to the new memory, or NULL on failure.

Definition at line 349 of file mkavl.c.

5.4.6.34 mkavl_rc_e mkavl_new (mkavl_tree_handle * *tree_h*, mkavl_compare_fn * *compare_fn_array*, size_t *compare_fn_array_count*, void * *context*, mkavl_allocator_st * *allocator*)

Create a new mkavl tree composed of AVL trees using the given array of comparison functions.

See also

mkavl_delete

Parameters

tree_h	A pointer to the memory location for the new tree.
compare_fn_array	An array of size compare_fn_array_count of the comparison functions for the mkavl. This allows the same set of data items to be stored in different orders by mulitple keys. A deep copy of the functions is made for the tree_h. There must be at least one function passed in.
compare_fn_array_count	The size of the compare_fn_array.
context	An opaque context passed back to the client in callbacks.
allocator	The memory allocation functions to use for the tree, or NULL if the default functions are to be used.

Returns

The return value

Definition at line 608 of file mkavl.c.

5.4.6.35 const char* mkavl_rc_e_get_string (mkavl_rc_e *rc*)

Get a string representation of the return code.

Parameters

rc	The return code

Returns

A string representation of the return code or "__Invalid__" if an invalid return code is input.

Definition at line 214 of file mkavl.c.

5.4.6.36 bool mkavl_rc_e_is_notok (mkavl_rc_e *rc*)

Indicates whether the return code is not in error.

Parameters

rc	The return code to check

Returns

true if the return code is not in error.

Definition at line 177 of file mkavl.c.

5.4.6.37 bool mkavl_rc_e_is_ok (mkavl_rc_e *rc*)

Indicates whether the return code is in error.

Parameters

rc	The return code to check

Returns

true if the return code is not in error.

Definition at line 189 of file mkavl.c.

5.4.6.38 bool mkavl_rc_e_is_valid (mkavl_rc_e *rc*)

Indicates whether the return code is valid.

Parameters

rc	The return code to check

Returns

true if the return code is valid.

Definition at line 201 of file mkavl.c.

5.4.6.39 mkavl_rc_e mkavl_remove (mkavl_tree_handle *tree_h*, const void * *item_to_remove*, void ** *found_item*)

Remove an item from mkavl tree.

Parameters

tree_h	The tree from which to remove.
item_to_remove	The item being removed.
found_item	If the item existed, the item that was found and removed.

Returns

The return code

Definition at line 1352 of file mkavl.c.

5.4.6.40 mkavl_rc_e mkavl_remove_key_idx (mkavl_tree_handle *tree_h*, size_t *key_idx*, const void * *item_to_remove*, void ** *found_item*)

Remove an item only from a specific AVL tree within the mkavl tree. Note that this must be used very carefully in conjunction with mkavl_add_key_idx to make sure the mkavl does not become out of sync. In steady state, the mkavl should always have an equal number of data items in each AVL tree. However, if a field changes in an item that affects some keys but not other, these APIs may be used to remove the item with the old values from the affected AVLs, update the values, and then re-add to each AVL tree from which the old item was removed.

See also

mkavl_add_key_idx

Parameters

tree_h	The mkavl tree.
key_idx	The index of the AVL tree to which the item is added.
item_to_remove	The item being removed.
found_item	If the item existed, the item that was found and removed.

Returns

The return code

Definition at line 1464 of file mkavl.c.

5.4.6.41 static bool mkavl_tree_is_valid (mkavl_tree_handle *tree_h*) [static]

Sanity check for mkavl tree objects.

Parameters

tree_h	The object to check. If NULL, false is returned.

Returns

true if the object is valid.

Definition at line 523 of file mkavl.c.

5.4.6.42 mkavl_rc_e mkavl_walk (mkavl_tree_handle *tree_h,* mkavl_walk_cb_fn *cb_fn,* void * *walk_context*)

Walk every node in the tree, calling the given function for each item. There is no guarantee on the order of the walk.

Parameters

tree_h	The tree to walk.
cb_fn	The callback function to apply to each item.
walk_context	The opaque walk context passed to the callback.

Returns

The return code.

Definition at line 1521 of file mkavl.c.

5.4.7 Variable Documentation

5.4.7.1 mkavl_allocator_st mkavl_allocator_default [static]

Initial value:

```
 {
    mkavl_default_malloc_fn,
    mkavl_default_free_fn
}
```

By default, we'll just use malloc and free if the client passes nothing in.

Definition at line 406 of file mkavl.c.

5.4.7.2 struct libavl_allocator mkavl_allocator_wrapper [static]

Initial value:

```
 {
    mkavl_malloc_wrapper,
    mkavl_free_wrapper
}
```

Wrapper to convert AVL callback to client callback passed to mkavl.

Definition at line 414 of file mkavl.c.

5.4.7.3 const char* const mkavl_find_type_e_string[] [static]

Initial value:

```
 {
    "Invalid",
    "Equal",
    "Greater than",
    "Less than",
    "Greater than or equal",
    "Less than or equal",
    "Max type"
}
```

String representations of the return codes.

See also

mkavl_find_type_e

Definition at line 230 of file mkavl.c.

5.4.7.4 const char* const mkavl_rc_e_string[] [static]

Initial value:

```
 {
    "Invalid RC",
    "Success",
    "Invalid input",
    "No memory",
    "Out of sync",
    "Max RC"
}
```

String representations of the return codes.

See also

mkavl_rc_e

Definition at line 156 of file mkavl.c.

5.5 mkavl.h File Reference

```
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>
#include <string.h>
#include <assert.h>
```

Classes

- struct mkavl_allocator_st_

Typedefs

- typedef struct mkavl_tree_st_ * mkavl_tree_handle
- typedef struct mkavl_iterator_st_ * mkavl_iterator_handle
- typedef enum mkavl_rc_e_ mkavl_rc_e
- typedef enum mkavl_find_type_e_ mkavl_find_type_e
- typedef void *(* mkavl_malloc_fn)(size_t size, void *context)
- typedef void(* mkavl_free_fn)(void *ptr, void *context)
- typedef struct mkavl_allocator_st_ mkavl_allocator_st
- typedef int32_t(* mkavl_compare_fn)(const void *item1, const void *item2, void *context)
- typedef mkavl_rc_e(* mkavl_item_fn)(void *item, void *context)
- typedef mkavl_rc_e(* mkavl_delete_context_fn)(void *context)
- typedef void *(* mkavl_copy_fn)(void *item, void *context)
- typedef mkavl_rc_e(* mkavl_walk_cb_fn)(void *item, void *tree_context, void *walk_context, bool *stop_walk)

Enumerations

- enum mkavl_rc_e_ {

  MKAVL_RC_E_INVALID, MKAVL_RC_E_SUCCESS, MKAVL_RC_E_EINVAL, MKAVL_RC_E_ENOMEM,

  MKAVL_RC_E_EOOSYNC, MKAVL_RC_E_MAX }
- enum mkavl_find_type_e_ {

  MKAVL_FIND_TYPE_E_INVALID, MKAVL_FIND_TYPE_E_EQUAL, MKAVL_FIND_TYPE_E_FIRST = MKAVL_FIND_TYPE_E_EQUAL, MKAVL_FIND_TYPE_E_GT,

  MKAVL_FIND_TYPE_E_LT, MKAVL_FIND_TYPE_E_GE, MKAVL_FIND_TYPE_E_LE, MKAVL_FIND_TYPE_E_MAX }

Functions

- bool mkavl_rc_e_is_notok (mkavl_rc_e rc)
- bool mkavl_rc_e_is_ok (mkavl_rc_e rc)
- bool mkavl_rc_e_is_valid (mkavl_rc_e rc)
- const char * mkavl_rc_e_get_string (mkavl_rc_e rc)
- bool mkavl_find_type_e_is_valid (mkavl_find_type_e type)
- const char * mkavl_find_type_e_get_string (mkavl_find_type_e type)
- mkavl_rc_e mkavl_new (mkavl_tree_handle *tree_h, mkavl_compare_fn *compare_fn_array, size_t compare_fn_array_count, void *context, mkavl_allocator_st *allocator)
- void * mkavl_get_tree_context (mkavl_tree_handle tree_h)
- mkavl_rc_e mkavl_delete (mkavl_tree_handle *tree_h, mkavl_item_fn item_fn, mkavl_delete_context_fn delete_context_fn)
- mkavl_rc_e mkavl_copy (const mkavl_tree_handle source_tree_h, mkavl_tree_handle *new_tree_h, mkavl_copy_fn copy_fn, mkavl_item_fn item_fn, bool use_source_context, void *new_context, mkavl_delete_context_fn delete_context_fn, mkavl_allocator_st *allocator)
- mkavl_rc_e mkavl_add (mkavl_tree_handle tree_h, void *item_to_add, void **existing_item)
- mkavl_rc_e mkavl_find (mkavl_tree_handle tree_h, mkavl_find_type_e type, size_t key_idx, const void *lookup_item, void **found_item)
- mkavl_rc_e mkavl_remove (mkavl_tree_handle tree_h, const void *item_to_remove, void **found_item)
- mkavl_rc_e mkavl_add_key_idx (mkavl_tree_handle tree_h, size_t key_idx, void *item_to_add, void **existing_item)
- mkavl_rc_e mkavl_remove_key_idx (mkavl_tree_handle tree_h, size_t key_idx, const void *item_to_remove, void **found_item)
- uint32_t mkavl_count (mkavl_tree_handle tree_h)
- mkavl_rc_e mkavl_walk (mkavl_tree_handle tree_h, mkavl_walk_cb_fn cb_fn, void *walk_context)
- mkavl_rc_e mkavl_iter_new (mkavl_iterator_handle *iterator_h, mkavl_tree_handle tree_h, size_t key_idx)
- mkavl_rc_e mkavl_iter_delete (mkavl_iterator_handle *iterator_h)
- mkavl_rc_e mkavl_iter_first (mkavl_iterator_handle iterator_h, void **item)
- mkavl_rc_e mkavl_iter_last (mkavl_iterator_handle iterator_h, void **item)
- mkavl_rc_e mkavl_iter_find (mkavl_iterator_handle iterator_h, void *lookup_item, void **found_item)
- mkavl_rc_e mkavl_iter_next (mkavl_iterator_handle iterator_h, void **item)
- mkavl_rc_e mkavl_iter_prev (mkavl_iterator_handle iterator_h, void **item)
- mkavl_rc_e mkavl_iter_cur (mkavl_iterator_handle iterator_h, void **item)

5.5.1 Detailed Description

Author

Matt Miller <matt@matthewjmiller.net>

5.5.2 LICENSE

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

5.5.3 DESCRIPTION

This is the public interface for the multi-key AVL interface.

Definition in file mkavl.h.

5.5.4 Typedef Documentation

5.5.4.1 typedef struct mkavl_allocator_st_ mkavl_allocator_st

Specifies the allocator functions for an AVL tree.

5.5.4.2 typedef int32_t(∗ mkavl_compare_fn)(const void ∗item1, const void ∗item2, void ∗context)

Prototype for comparing two items. The context is what was passed in when the AVL tree was created.

Returns

Zero if item1 and item2 are equal, 1 if item1 is greater, and -1 if item2 is greater.

Definition at line 189 of file mkavl.h.

5.5.4.3 typedef void∗(∗ mkavl_copy_fn)(void ∗item, void ∗context)

Prototype for a function that is applied when copying items from an existing tree to a new tree.

Parameters

item	Item to be copied
context	Client context for the tree

Returns

A pointer to the item to be placed into the new tree.

Definition at line 222 of file mkavl.h.

5.5.4.4 typedef mkavl_rc_e(* mkavl_delete_context_fn)(void *context)

Prototype for a function that is a callback to the client to operate on its context when mkavl_delete() is called.

Parameters

context	The client context

Returns

The return code

Definition at line 211 of file mkavl.h.

5.5.4.5 typedef enum mkavl_find_type_e_ mkavl_find_type_e

The type of find for lookups.

5.5.4.6 typedef void(* mkavl_free_fn)(void *ptr, void *context)

Prototype for freeing items.

Definition at line 169 of file mkavl.h.

5.5.4.7 typedef mkavl_rc_e(* mkavl_item_fn)(void *item, void *context)

Prototype for a function that is applied to every element for various operations.

Parameters

item	The item on which to operate
context	The client context

Returns

The return code

Definition at line 201 of file mkavl.h.

5.5.4.8 typedef struct mkavl_iterator_st_* mkavl_iterator_handle

Opaque pointer to reference instances of AVL iterators

Definition at line 117 of file mkavl.h.

5.5.4.9 typedef void*(* mkavl_malloc_fn)(size_t size, void *context)

Prototype for allocating items.

Definition at line 163 of file mkavl.h.

5.5.4.10 typedef enum mkavl_rc_e_ mkavl_rc_e

Return codes used to indicate whether a function call was successful.

5.5.4.11 typedef struct mkavl_tree_st_* mkavl_tree_handle

Opaque pointer to reference instances of AVL trees

Definition at line 114 of file mkavl.h.

5.5.4.12 typedef mkavl_rc_e(* mkavl_walk_cb_fn)(void *item, void *tree_context, void *walk_context, bool *stop_walk)

Prototype for a function applied to items when walking over the entire tree.

Parameters

item	The current item on which to operate.
tree_context	The context for the tree.
walk_context	The context for the current walk.
stop_walk	Impelementor should set to true to stop the walk upon return.

Definition at line 234 of file mkavl.h.

5.5.5 Enumeration Type Documentation

5.5.5.1 enum mkavl_find_type_e_

The type of find for lookups.

Enumerator:

MKAVL_FIND_TYPE_E_INVALID Invalid type

MKAVL_FIND_TYPE_E_EQUAL Find an item equal to

MKAVL_FIND_TYPE_E_FIRST First valid find type

MKAVL_FIND_TYPE_E_GT Find an item greater than

MKAVL_FIND_TYPE_E_LT Find an item less than

MKAVL_FIND_TYPE_E_GE Find an item greater than or equal to

MKAVL_FIND_TYPE_E_LE Find an item less than or equal to

MKAVL_FIND_TYPE_E_MAX Max value for bounds testing

Definition at line 140 of file mkavl.h.

5.5.5.2 enum mkavl_rc_e_

Return codes used to indicate whether a function call was successful.

Enumerator:

MKAVL_RC_E_INVALID Invalid return code, should never be used

MKAVL_RC_E_SUCCESS Successful return

MKAVL_RC_E_EINVAL Function received an invalid input

MKAVL_RC_E_ENOMEM Function failed to allocate memory

MKAVL_RC_E_EOOSYNC Internal data structures are out of sync

MKAVL_RC_E_MAX Max return code for bounds testing

Definition at line 122 of file mkavl.h.

5.5.6 Function Documentation

5.5.6.1 mkavl_rc_e mkavl_add (mkavl_tree_handle *tree_h,* void * *item_to_add,* void ** *existing_item*)

Add an item to each AVL tree in the mkavl.

Parameters

tree_h	The mkavl tree.
item_to_add	A pointer to the data to add.
existing_item	If an existing item is found, it is returned. Otherwise, NULL if the item was not found.

Returns

The return code

Definition at line 1004 of file mkavl.c.

5.5.6.2 mkavl_rc_e mkavl_add_key_idx (mkavl_tree_handle *tree_h,* size_t *key_idx,* void * *item_to_add,* void ** *existing_item*)

Add an item only to a specific AVL tree within the mkavl tree. Note that this must be used very carefully in conjunction with mkavl_remove_key_idx to make sure the mkavl does not become out of sync. In steady state, the mkavl should always have an equal number of data items in each AVL tree. However, if a field changes in an item that

affects some keys but not other, these APIs may be used to remove the item with the old values from the affected AVLs, update the values, and then re-add to each AVL tree from which the old item was removed.

See also

mkavl_remove_key_idx

Parameters

tree_h	The mkavl tree.
key_idx	The index of the AVL tree to which the item is added.
item_to_add	The item being added.
existing_item	If an existing item is found, it is returned.

Returns

The return code

Definition at line 1422 of file mkavl.c.

5.5.6.3 mkavl_rc_e mkavl_copy (mkavl_tree_handle *source_tree_h*, mkavl_tree_handle * *new_tree_h*, mkavl_copy_fn *copy_fn*, mkavl_item_fn *item_fn*, bool *use_source_context*, void * *new_context*, mkavl_delete_context_fn *delete_context_fn*, mkavl_allocator_st * *allocator*)

Deep copy a mkavl tree into a new tree.

Parameters

source_tree_h	The tree from which to copy.
new_tree_h	A pointer to the new tree to which the copy will be done.
copy_fn	A function that is applied to each item in the source tree before it is copied to the new tree. This is applied once per item regardless of how many AVL trees there are. E.g., this may allocate a deep copy of the data. If NULL, then a shallow copy of the data is copied to the new tree. Note that this function is called with the source tree's context value.
item_fn	If there is an error copying the new tree, this function is applied to all items in the new tree as it is destroyed. E.g., this may be a function to free the data.
use_source_context	If true, the client context from source_tree_h is used for the new tree. If false, new_context is used for the new tree.
new_context	The opaque client context to use for the new tree if user_source_context if false.
delete_context_fn	Upon an error, this function will be applied to the new_context if use_source_context is false. If use_source_context is true, this is a no-op in the case of an error (will not call on the source tree's context). If NULL, no function is applied. If given, the function will be called even if the client context is NULL.
allocator	The memory allocation functions to use for the new tree.

Returns

The return code

Definition at line 862 of file mkavl.c.

5.5.6.4 uint32_t mkavl_count (mkavl_tree_handle *tree_h*)

Get a count of the total number of items within the tree. Note that this is the steady state account, i.e., broadly, the number of calls to mkavl_add that succeeded minus the number of calls to mkavl_remove that succeeded. If this is called in between calls to mkavl_add_key_idx and mkavl_remove_key_idx, then it is possible for individual AVL trees to have a different count.

See also

mkavl_add
mkavl_remove
mkavl_add_key_idx
mkavl_remove_key_idx

Definition at line 1502 of file mkavl.c.

5.5.6.5 mkavl_rc_e mkavl_delete (mkavl_tree_handle * *tree_h*, mkavl_item_fn *item_fn*, mkavl_delete_context_fn *delete_context_fn*)

The destroys the tree that was allocated by mkavl_new. Note that this doesn't actually free the data of the items as that is left to the client. The item_fn parameter can be used for this purpose. Upon return, the tree_h memory is set to NULL. This is O(N lg N) for N items.

See also

mkavl_new
mkavl_delete_tree

Parameters

tree_h	A pointer the the tree to free.
item_fn	This function is applied to each item after it has been removed from all the AVL trees. This is only called once per item regardless of how many AVL trees there are. E.g., this could be the function to free the memory for the data. If NULL, no function is applied.
delete_context_fn	This function is applied to the tree's opaque client context that was given via mkavl_new(). E.g., this could free the client context if it was a heap memory pointer. If NULL, no function is applied. If given, the function will be called even if the client context is NULL. This function is called after all the items have been deleted and item functions called.

Returns

The return code

Definition at line 750 of file mkavl.c.

5.5.6.6 mkavl_rc_e mkavl_find (mkavl_tree_handle *tree_h,* mkavl_find_type_e *type,* size_t *key_idx,* const void * *lookup_item,* void ** *found_item*)

Find an item in the tree.

Parameters

tree_h	The tree to search
type	The type of lookup to do.
key_idx	The AVL tree being searched.
lookup_item	The item to use as the lookup target.
found_item	The item found for the lookup.

Returns

The return code

Definition at line 1275 of file mkavl.c.

5.5.6.7 const char* mkavl_find_type_e_get_string (mkavl_find_type_e *type*)

Get a string representation of the find type.

Parameters

type	The find type

Returns

A string representation of the return code or "__Invalid__" if an invalid return code is input.

Definition at line 266 of file mkavl.c.

5.5.6.8 bool mkavl_find_type_e_is_valid (mkavl_find_type_e *type*)

Indicates whether the return code is valid.

Parameters

type	The type to check

Returns

true if the return code is valid.

Definition at line 252 of file mkavl.c.

5.5.6.9 void* mkavl_get_tree_context (mkavl_tree_handle *tree_h*)

Get the context pointer for the tree.

See also

mkavl_new

Parameters

tree_h	The tree whose context to get. This must be a valid, non-NULL tree pointer or else a crash will occur.

Returns

The tree's context (possibly NULL if that was passed to mkavl_new()).

Definition at line 721 of file mkavl.c.

5.5.6.10 mkavl_rc_e mkavl_iter_cur (mkavl_iterator_handle *iterator_h*, void ** *item*)

Get the current item in the iteration.

See also

mkavl_iter_new

Parameters

iterator_h	The iterator to use.
item	The item found.

Returns

The return code

Definition at line 1780 of file mkavl.c.

5.5.6.11 mkavl_rc_e mkavl_iter_delete (mkavl_iterator_handle * *iterator_h*)

Destroy the iterator.

See also

mkavl_iter_new

Parameters

iterator_h	The iterator to free. Upon return, this will be set to NULL.

Returns

The return code

Definition at line 1617 of file mkavl.c.

5.5.6.12 mkavl_rc_e mkavl_iter_find (mkavl_iterator_handle *iterator_h*, void * *lookup_item*, void ** *found_item*)

Find an item and update the iterator to that node.

See also

mkavl_iter_new

Parameters

iterator_h	The iterator to use.
lookup_item	The item to find.
found_item	The item found.

Returns

The return code

Definition at line 1701 of file mkavl.c.

5.5.6.13 mkavl_rc_e mkavl_iter_first (mkavl_iterator_handle *iterator_h*, void ** *item*)

Get the first item in the iteration.

See also

mkavl_iter_new

Parameters

iterator_h	The iterator to use.
item	The item found.

Returns

The return code

Definition at line 1648 of file mkavl.c.

5.5.6.14 mkavl_rc_e mkavl_iter_last (mkavl_iterator_handle *iterator_h*, void ** *item*)

Get the last item in the iteration.

See also

mkavl_iter_new

Parameters

iterator_h	The iterator to use.
item	The item found.

Returns

The return code

Definition at line 1674 of file mkavl.c.

5.5.6.15 mkavl_rc_e mkavl_iter_new (mkavl_iterator_handle * *iterator_h*, mkavl_tree_handle *tree_h*, size_t *key_idx*)

Create a new iterator to use.

See also

mkavl_iter_delete

Parameters

iterator_h	The pointer to fill in with the new iterator.
tree_h	The tree on which to iterate.
key_idx	The index of the AVL tree on which to iterate.

Returns

The return code

Definition at line 1565 of file mkavl.c.

5.5.6.16 mkavl_rc_e mkavl_iter_next (mkavl_iterator_handle *iterator_h*, void ** *item*)

Get the next item in the iteration and update the iterator to that node.

See also

mkavl_iter_new

Parameters

iterator_h	The iterator to use.
item	The item found.

Returns

The return code

Definition at line 1730 of file mkavl.c.

5.5.6.17 mkavl_rc_e mkavl_iter_prev (mkavl_iterator_handle *iterator_h*, void ** *item*)

Get the previous item in the iteration and update the iterator to that node.

See also

mkavl_iter_new

Parameters

iterator_h	The iterator to use.
item	The item found.

Returns

The return code

Definition at line 1755 of file mkavl.c.

5.5.6.18 mkavl_rc_e mkavl_new (mkavl_tree_handle * *tree_h*, mkavl_compare_fn * *compare_fn_array*, size_t *compare_fn_array_count*, void * *context*, mkavl_allocator_st * *allocator*)

Create a new mkavl tree composed of AVL trees using the given array of comparison functions.

See also

mkavl_delete

Parameters

tree_h	A pointer to the memory location for the new tree.
compare_fn_array	An array of size compare_fn_array_count of the comparison functions for the mkavl. This allows the same set of data items to be stored in different orders by mulitple keys. A deep copy of the functions is made for the tree_h. There must be at least one function passed in.
compare_fn_array_count	The size of the compare_fn_array.
context	An opaque context passed back to the client in callbacks.
allocator	The memory allocation functions to use for the tree, or NULL if the default functions are to be used.

Returns

The return value

Definition at line 608 of file mkavl.c.

5.5.6.19 const char* mkavl_rc_e_get_string (mkavl_rc_e *rc*)

Get a string representation of the return code.

Parameters

rc	The return code

Returns

A string representation of the return code or "__Invalid__" if an invalid return code is input.

Definition at line 214 of file mkavl.c.

5.5.6.20 bool mkavl_rc_e_is_notok (mkavl_rc_e *rc*)

Indicates whether the return code is not in error.

Parameters

rc	The return code to check

Returns

true if the return code is not in error.

Definition at line 177 of file mkavl.c.

5.5.6.21 bool mkavl_rc_e_is_ok (mkavl_rc_e *rc*)

Indicates whether the return code is in error.

Parameters

rc	The return code to check

Returns

true if the return code is not in error.

Definition at line 189 of file mkavl.c.

5.5.6.22 bool mkavl_rc_e_is_valid (mkavl_rc_e *rc*)

Indicates whether the return code is valid.

Parameters

rc	The return code to check

Returns

true if the return code is valid.

Definition at line 201 of file mkavl.c.

5.5.6.23 mkavl_rc_e mkavl_remove (mkavl_tree_handle *tree_h*, const void * *item_to_remove*, void ** *found_item*)

Remove an item from mkavl tree.

Parameters

tree_h	The tree from which to remove.
item_to_remove	The item being removed.
found_item	If the item existed, the item that was found and removed.

Returns

The return code

Definition at line 1352 of file mkavl.c.

5.5.6.24 mkavl_rc_e mkavl_remove_key_idx (mkavl_tree_handle *tree_h*, size_t *key_idx*, const void * *item_to_remove*, void ** *found_item*)

Remove an item only from a specific AVL tree within the mkavl tree. Note that this must be used very carefully in conjunction with mkavl_add_key_idx to make sure the mkavl does not become out of sync. In steady state, the mkavl should always have an equal number of data items in each AVL tree. However, if a field changes in an item that affects some keys but not other, these APIs may be used to remove the item with the old values from the affected AVLs, update the values, and then re-add to each AVL tree from which the old item was removed.

See also

mkavl_add_key_idx

Parameters

tree_h	The mkavl tree.
key_idx	The index of the AVL tree to which the item is added.

item_to_remove	The item being removed.
found_item	If the item existed, the item that was found and removed.

Returns

The return code

Definition at line 1464 of file mkavl.c.

5.5.6.25 mkavl_rc_e mkavl_walk (mkavl_tree_handle *tree_h*, mkavl_walk_cb_fn *cb_fn*, void * *walk_context*)

Walk every node in the tree, calling the given function for each item. There is no guarantee on the order of the walk.

Parameters

tree_h	The tree to walk.
cb_fn	The callback function to apply to each item.
walk_context	The opaque walk context passed to the callback.

Returns

The return code.

Definition at line 1521 of file mkavl.c.

5.6 test/test_mkavl.c File Reference

```
#include <stdlib.h>
#include <stdbool.h>
#include <unistd.h>
#include <errno.h>
#include <time.h>
#include <stdio.h>
#include "../mkavl.h"
```

Classes

- struct test_mkavl_opts_st_
- struct mkavl_test_input_st_
- struct mkavl_test_ctx_st_
- struct mkavl_test_walk_ctx_st_

Defines

- #define LOG_FAIL(fmt, args...)
- #define NELEMS(x) (sizeof(x) / sizeof(x[0]))
- #define CT_ASSERT(e) extern char (∗CT_ASSERT(void)) [sizeof(char[1 - 2∗!(e)])]
- #define MKAVL_TEST_RUNAWAY_SANITY 100000
- #define MKAVL_TEST_MAGIC 0x1234ABCD

Typedefs

- typedef struct test_mkavl_opts_st_ test_mkavl_opts_st
- typedef struct mkavl_test_input_st_ mkavl_test_input_st
- typedef struct mkavl_test_ctx_st_ mkavl_test_ctx_st
- typedef enum mkavl_test_key_e_ mkavl_test_key_e
- typedef struct mkavl_test_walk_ctx_st_ mkavl_test_walk_ctx_st

Enumerations

- enum mkavl_test_key_e_ { MKAVL_TEST_KEY_E_ASC, MKAVL_TEST_KEY_-E_DESC, MKAVL_TEST_KEY_E_MAX }

Functions

- static void print_usage (bool do_exit, int32_t exit_val)
- static void print_opts (test_mkavl_opts_st ∗opts)
- static void parse_command_line (int argc, char ∗∗argv, test_mkavl_opts_st ∗opts)
- static void permute_array (const uint32_t ∗src_array, uint32_t ∗dst_array, size_t num_elem)
- static int uint32_t_cmp (const void ∗a, const void ∗b)
- static uint32_t get_unique_count (uint32_t ∗array, size_t num_elem)
- static bool run_mkavl_test (mkavl_test_input_st ∗input)
- int main (int argc, char ∗argv[])
- static void ∗ mkavl_test_copy_malloc (size_t size, void ∗context)
- static void mkavl_test_copy_free (void ∗ptr, void ∗context)
- static int32_t mkavl_cmp_fn1 (const void ∗item1, const void ∗item2, void ∗context)
- static int32_t mkavl_cmp_fn2 (const void ∗item1, const void ∗item2, void ∗context)
- static bool mkavl_test_new_error (void)
- static bool mkavl_test_new (mkavl_test_input_st ∗input, mkavl_allocator_st ∗allocator)
- static mkavl_rc_e mkavl_test_delete_context (void ∗context)
- static bool mkavl_test_delete (mkavl_test_input_st ∗input, mkavl_item_fn item_-fn, mkavl_delete_context_fn delete_context_fn, mkavl_delete_context_fn delete_-copy_context_fn)
- static bool mkavl_test_add_error (mkavl_test_input_st ∗input)
- static bool mkavl_test_add (mkavl_test_input_st ∗input)
- static uint32_t ∗ mkavl_test_find_val (mkavl_test_input_st ∗input, uint32_t val, mkavl_find_type_e type)

- static bool mkavl_test_find (mkavl_test_input_st *input, mkavl_find_type_e type)
- static bool mkavl_test_find_error (mkavl_test_input_st *input)
- static bool mkavl_test_add_remove_key (mkavl_test_input_st *input)
- static bool mkavl_test_add_key_error (mkavl_test_input_st *input)
- static bool mkavl_test_remove_key_error (mkavl_test_input_st *input)
- static void * mkavl_test_copy_fn (void *item, void *context)
- static bool mkavl_test_copy (mkavl_test_input_st *input)
- static bool mkavl_test_iterator (mkavl_test_input_st *input)
- static mkavl_rc_e mkavl_test_walk_cb (void *item, void *tree_context, void *walk_context, bool *stop_walk)
- static bool mkavl_test_walk (mkavl_test_input_st *input)
- static bool mkavl_test_remove (mkavl_test_input_st *input)
- static mkavl_rc_e mkavl_test_item_fn (void *item, void *context)

Variables

- static const uint32_t default_node_cnt = 15
- static const uint32_t default_run_cnt = 15
- static const uint8_t default_verbosity = 0
- static const uint32_t default_range_start = 0
- static const uint32_t default_range_end = 100
- static mkavl_allocator_st copy_allocator
- static const mkavl_test_key_e mkavl_key_opposite []
- static mkavl_compare_fn cmp_fn_array [] = { mkavl_cmp_fn1 , mkavl_cmp_fn2 }
- static const mkavl_find_type_e mkavl_key_find_type [MKAVL_TEST_KEY_E_MAX][(MKAVL_FIND_TYPE_E_MAX+1)]

5.6.1 Detailed Description

Author

Matt Miller <matt@matthewjmiller.net>

5.6.2 LICENSE

5.6.3 DESCRIPTION

Unit test for mkavl library.

```
    Test the mkavl structure

    Usage:
    -s <seed>
       The starting seed for the RNG (default=seeded by time()).
    -n <nodes>
       The number of nodes to place in the trees (default=15).
    -b <range beginning>
       The smallest (inclusive ) possible data value in the range of values
       (default=0).
    -e <range ending>
       The largest (exclusive) possible data value in the range of values
       (default=100).
    -r <runs>
       The number of runs to do (default=15).
    -v <verbosity level>
       A higher number gives more output (default=0).
   -h
      Display this help message.
```

Definition in file test_mkavl.c.

5.6.4 Define Documentation

5.6.4.1 #define CT_ASSERT(*e*) extern char (*CT_ASSERT(void)) [sizeof(char[1 - 2*!(e)])]

Compile time assert macro from: http://www.pixelbeat.org/programming/gcc/static_assert.html

Definition at line 75 of file test_mkavl.c.

5.6.4.2 #define LOG_FAIL(*fmt, args...*)

Value:

```
do { \
    printf("FAILURE(%s:%u): " fmt "\n", __FUNCTION__, __LINE__, ##args); \
} while (0)
```

Display a failure message.

Definition at line 58 of file test_mkavl.c.

5.6.4.3 #define MKAVL_TEST_MAGIC 0x1234ABCD

Magic value for sanity checks

Definition at line 451 of file test_mkavl.c.

5.6.4.4 #define MKAVL_TEST_RUNAWAY_SANITY 100000

Sanity check for infinite loops.

Definition at line 81 of file test_mkavl.c.

5.6.4.5 #define NELEMS(*x*) (sizeof(x) / sizeof(x[0]))

Determine the number of elements in an array.

Definition at line 67 of file test_mkavl.c.

5.6.5 Typedef Documentation

5.6.5.1 typedef struct mkavl_test_ctx_st_ mkavl_test_ctx_st

The context storted for a tree.

5.6.5.2 typedef struct mkavl_test_input_st_ mkavl_test_input_st

The input structure to pass test parameters to functions.

5.6.5.3 typedef enum mkavl_test_key_e_ mkavl_test_key_e

The values for the key ordering.

5.6.5.4 typedef struct mkavl_test_walk_ctx_st_ mkavl_test_walk_ctx_st

The context for mkavl_walk()

5.6.5.5 typedef struct test_mkavl_opts_st_ test_mkavl_opts_st

State for the current test execution.

5.6.6 Enumeration Type Documentation

5.6.6.1 enum mkavl_test_key_e_

The values for the key ordering.

Enumerator:

MKAVL_TEST_KEY_E_ASC Ascending order

MKAVL_TEST_KEY_E_DESC Descending order

MKAVL_TEST_KEY_E_MAX Max value for boundary testing

Definition at line 582 of file test_mkavl.c.

5.6.7 Function Documentation

5.6.7.1 static uint32_t get_unique_count (uint32_t * *array*, size_t *num_elem*) [static]

Get a count of the number of unique items in the array. This assumes the input array is sorted.

Parameters

array	The sorted array.
num_elem	The number of elements in the array.

Returns

The unique number of elements in the array.

Definition at line 317 of file test_mkavl.c.

5.6.7.2 int main (int *argc*, char * *argv[]*)

Main function to test objects.

Definition at line 370 of file test_mkavl.c.

5.6.7.3 static int32_t mkavl_cmp_fn1 (const void * *item1*, const void * *item2*, void * *context*) [static]

Simply compare the uint32_t values.

Parameters

item1	Item to compare
item2	Item to compare
context	Context for the tree

Returns

Comparison result

Definition at line 528 of file test_mkavl.c.

5.6.7.4 static int32_t mkavl_cmp_fn2 (const void * *item1*, const void * *item2*, void * *context*) [static]

Reverse the values of the items.

Parameters

item1	Item to compare
item2	Item to compare
context	Context for the tree

Returns

Comparison result

Definition at line 558 of file test_mkavl.c.

5.6.7.5 static bool mkavl_test_add (mkavl_test_input_st * *input*) [static]

Test mkavl_add().

Parameters

input	The input state for the test.

Returns

True if test passed.

Definition at line 788 of file test_mkavl.c.

5.6.7.6 static bool mkavl_test_add_error (mkavl_test_input_st * *input*) [static]

Test mkavl_add() for error handling.

Parameters

input	The input state for the test.

Returns

True if test passed.

Definition at line 753 of file test_mkavl.c.

5.6.7.7 static bool mkavl_test_add_key_error (mkavl_test_input_st * *input*) [static]

Test mkavl_add_key() for error handling.

Parameters

input	The input state for the test.

Returns

True if test passed.

Definition at line 1214 of file test_mkavl.c.

5.6.7.8 static bool mkavl_test_add_remove_key (mkavl_test_input_st * input) [static]

Test mkavl_remove_key() and mkavl_add_key().

Parameters

input	The input state for the test.

Returns

True if test passed.

Definition at line 1110 of file test_mkavl.c.

5.6.7.9 static bool mkavl_test_copy (mkavl_test_input_st * input) [static]

Test mkavl_copy().

Parameters

input	The input state for the test.

Returns

True if test passed.

Definition at line 1330 of file test_mkavl.c.

5.6.7.10 static void* mkavl_test_copy_fn (void * item, void * context) [static]

The callback for mkavl_copy().

Parameters

item	The original item being copied.
context	The context for the tree.

Returns

The item to place in the copied tree.

Definition at line 1309 of file test_mkavl.c.

5.6.7.11 static void mkavl_test_copy_free (void * ptr, void * context) [static]

The free function for the copied tree.

Parameters

ptr	The memory to free.
context	The tree context.

Definition at line 498 of file test_mkavl.c.

5.6.7.12 static void* mkavl_test_copy_malloc (size_t *size*, void * *context*) [static]

The malloc function for the copied tree.

Parameters

size	Size of memory to allocate.
context	The tree context.

Returns

A pointer to the memory or NULL if allocation was not possible.

Definition at line 477 of file test_mkavl.c.

5.6.7.13 static bool mkavl_test_delete (mkavl_test_input_st * *input*, mkavl_item_fn *item_fn*, mkavl_delete_context_fn *delete_context_fn*, mkavl_delete_context_fn *delete_copy_context_fn*) [static]

Test mkavl_delete().

Parameters

input	The input state for the test.
item_fn	The function to apply to deleted items.
delete_context_fn	The function to apply to the tree's context.
delete_copy_context_fn	The function to apply to the copied tree's context.

Returns

True if test passed.

Definition at line 720 of file test_mkavl.c.

5.6.7.14 static mkavl_rc_e mkavl_test_delete_context (void * *context*) [static]

The callback for freeing the context memory.

Parameters

context	The context to free.

Returns

The return code

Definition at line 695 of file test_mkavl.c.

5.6.7.15 static bool mkavl_test_find (mkavl_test_input_st * *input*, mkavl_find_type_e *type*) [static]

Test mkavl_find().

Parameters

input	The input state for the test.
type	The type of find.

Returns

True if test passed.

Definition at line 938 of file test_mkavl.c.

5.6.7.16 static bool mkavl_test_find_error (mkavl_test_input_st * *input*) [static]

Test mkavl_find() for error handling.

Parameters

input	The input state for the test.

Returns

True if test passed.

Definition at line 1058 of file test_mkavl.c.

5.6.7.17 static uint32_t* mkavl_test_find_val (mkavl_test_input_st * *input*, uint32_t *val*, mkavl_find_type_e *type*) [static]

Test mkavl_find().

Parameters

input	The input state for the test.
val	The value on which to search.
type	The type of find.

Returns

True if test passed.

Definition at line 832 of file test_mkavl.c.

5.6.7.18 static mkavl_rc_e mkavl_test_item_fn (void * *item*, void * *context*) [static]

The callback for per-item functions.

Parameters

item	The current item.
context	The tree's context.

Returns

The return code

Definition at line 1743 of file test_mkavl.c.

5.6.7.19 static bool mkavl_test_iterator (mkavl_test_input_st * *input*) [static]

Test mkavl iterators.

Parameters

input	The input state for the test.

Returns

True if test passed.

Definition at line 1380 of file test_mkavl.c.

5.6.7.20 static bool mkavl_test_new (mkavl_test_input_st * *input*, mkavl_allocator_st * *allocator*) [static]

Test mkavl_new().

Parameters

input	The input state for the test.
allocator	The allocator to use.

Returns

True if test passed.

Definition at line 666 of file test_mkavl.c.

5.6.7.21 static bool mkavl_test_new_error (void) [static]

Test mkavl_new() for error handling.

Returns

True if test passed.

Definition at line 625 of file test_mkavl.c.

5.6.7.22 static bool mkavl_test_remove (mkavl_test_input_st * input) [static]

Test mkavl_remove().

Parameters

input	The input state for the test.

Returns

True if test passed.

Definition at line 1700 of file test_mkavl.c.

5.6.7.23 static bool mkavl_test_remove_key_error (mkavl_test_input_st * input) [static]

Test mkavl_remove_key() for error handling.

Parameters

input	The input state for the test.

Returns

True if test passed.

Definition at line 1261 of file test_mkavl.c.

5.6.7.24 static bool mkavl_test_walk (mkavl_test_input_st * input) [static]

Test mkavl_walk().

Parameters

input	The input state for the test.

Returns

True if test passed.

Definition at line 1653 of file test_mkavl.c.

5.6.7.25 static mkavl_rc_e mkavl_test_walk_cb (void * *item*, void * *tree_context*, void * *walk_context*, bool * *stop_walk*) [static]

The callback for mkavl_walk().

Parameters

item	The current item in the walk.
tree_context	The tree's context.
walk_context	The walks' context.
stop_walk	Can be set true to stop the walk upon return.

Returns

The return code

Definition at line 1625 of file test_mkavl.c.

5.6.7.26 static void parse_command_line (int *argc*, char ** *argv*, test_mkavl_opts_st * *opts*) [static]

Store the command line options into a local structure.

Parameters

argc	The number of options
argv	The string for the options.
opts	The local structure in which to store the parsed info.

Definition at line 175 of file test_mkavl.c.

5.6.7.27 static void permute_array (const uint32_t * *src_array*, uint32_t * *dst_array*, size_t *num_elem*) [static]

Create a permutation of the given array.

Parameters

src_array	The array to permute.
dst_array	The output location of the permuted array.
num_elem	The number of array elements in the arrays.

Definition at line 267 of file test_mkavl.c.

5.6.7.28 static void print_opts (test_mkavl_opts_st * opts) [static]

Utility function to output the value of the options.

Parameters

opts	The options to output.

Definition at line 155 of file test_mkavl.c.

5.6.7.29 static void print_usage (bool do_exit, int32_t exit_val) [static]

Display the program's help screen and exit as needed.

Parameters

do_exit	Whether to exit after the output.
exit_val	If exiting the value with which to exit.

Definition at line 119 of file test_mkavl.c.

5.6.7.30 static bool run_mkavl_test (mkavl_test_input_st * input) [static]

Runs all of the tests.

Parameters

input	The input state for the test.

Returns

True if test passed.

Definition at line 1764 of file test_mkavl.c.

5.6.7.31 static int uint32_t_cmp (const void * a, const void * b) [static]

Callback to compare two pointers to uint32_t's.

Parameters

a	An item to compare.
b	The other item to compare.

Returns

0 if equal, -1 if a $<$ b, and 1 if a $>$ b

Definition at line 294 of file test_mkavl.c.

5.6.8 Variable Documentation

5.6.8.1 mkavl_compare_fn cmp_fn_array[] = { mkavl_cmp_fn1 , mkavl_cmp_fn2 } [static]

The comparison functions to use

Definition at line 598 of file test_mkavl.c.

5.6.8.2 mkavl_allocator_st copy_allocator [static]

Initial value:

```
 {
    mkavl_test_copy_malloc,
    mkavl_test_copy_free
}
```

Allocators to use for the copied tree.

Definition at line 514 of file test_mkavl.c.

5.6.8.3 const uint32_t default_node_cnt = 15 [static]

The default node count for runs

Definition at line 84 of file test_mkavl.c.

5.6.8.4 const uint32_t default_range_end = 100 [static]

The default end of the range for node data values

Definition at line 92 of file test_mkavl.c.

5.6.8.5 const uint32_t default_range_start = 0 [static]

The default start of the range for node data values

Definition at line 90 of file test_mkavl.c.

5.6.8.6 const uint32_t default_run_cnt = 15 [static]

The default number of test runs

Definition at line 86 of file test_mkavl.c.

5.6.8.7 const uint8_t default_verbosity = 0 [static]

The default verbosity level of messages displayed

Definition at line 88 of file test_mkavl.c.

5.6.8.8 const mkavl_find_type_e mkavl_key_find_type[MKAVL_TEST_KEY_E_MAX][(MKAVL_FIND_TYPE_E_MAX+1)] [static]

Initial value:

```
 {
    { MKAVL_FIND_TYPE_E_INVALID, MKAVL_FIND_TYPE_E_EQUAL,
      MKAVL_FIND_TYPE_E_GT, MKAVL_FIND_TYPE_E_LT,
      MKAVL_FIND_TYPE_E_GE, MKAVL_FIND_TYPE_E_LE,
      MKAVL_FIND_TYPE_E_MAX },
    { MKAVL_FIND_TYPE_E_INVALID, MKAVL_FIND_TYPE_E_EQUAL,
      MKAVL_FIND_TYPE_E_LT, MKAVL_FIND_TYPE_E_GT,
      MKAVL_FIND_TYPE_E_LE, MKAVL_FIND_TYPE_E_GE,
      MKAVL_FIND_TYPE_E_MAX }
}
```

The type to use for find lookups for the key types.

Definition at line 609 of file test_mkavl.c.

5.6.8.9 const mkavl_test_key_e mkavl_key_opposite[] [static]

Initial value:

```
 {
    MKAVL_TEST_KEY_E_DESC,
    MKAVL_TEST_KEY_E_ASC,
}
```

The opposite key, used for certain find operations

Definition at line 592 of file test_mkavl.c.