NAME

ARB_PROTOTYPE, **ARB_PROTOTYPE_STATIC**, **ARB_PROTOTYPE_INSERT**, **ARB_PROTOTYPE_INSERT_COLOR**, **ARB_PROTOTYPE_REMOVE**, **ARB_PROTOTYPE_REMOVE_COLOR**, **ARB_PROTOTYPE_FIND**, **ARB_PROTOTYPE_NFIND**, **ARB_PROTOTYPE_NEXT**, **ARB_PROTOTYPE_PREV**, **ARB_PROTOTYPE_MINMAX**, **ARB_PROTOTYPE_REINSERT**, **ARB_GENERATE**, **ARB_GENERATE_STATIC**, **ARB_GENERATE_INSERT**, **ARB_GENERATE_INSERT_COLOR**, **ARB_GENERATE_REMOVE**, **ARB_GENERATE_REMOVE_COLOR**, **ARB_GENERATE_FIND**, **ARB_GENERATE_NFIND**, **ARB_GENERATE_NEXT**, **ARB_GENERATE_PREV**, **ARB_GENERATE_MINMAX**, **ARB_GENERATE_REINSERT**, **ARB8_ENTRY**, **ARB16_ENTRY**, **ARB32_ENTRY**, **ARB8_HEAD**, **ARB16_HEAD**, **ARB32_HEAD**, **ARB_ALLOCSIZE**, **ARB_INITIALIZER**, **ARB_ROOT**, **ARB_EMPTY**, **ARB_FULL**, **ARB_CURNODES**, **ARB_MAXNODES**, **ARB_NEXT**, **ARB_PREV**, **ARB_MIN**, **ARB_MAX**, **ARB_FIND**, **ARB_NFIND**, **ARB_LEFT**, **ARB_LEFTIDX**, **ARB_RIGHT**, **ARB_RIGHTIDX**, **ARB_PARENT**, **ARB_PARENTIDX**, **ARB_GETFREE**, **ARB_FREEIDX**, **ARB_FOREACH**, **ARB_FOREACH_FROM**, **ARB_FOREACH_SAFE**, **ARB_FOREACH_REVERSE**, **ARB_FOREACH_REVERSE_FROM**, **ARB_FOREACH_REVERSE_SAFE**, **ARB_INIT**, **ARB_INSERT**, **ARB_REMOVE**, **ARB_REINSERT**, **ARB_RESET_TREE** - array-based red-black trees

SYNOPSIS

#include <sys/arb.h>

ARB_PROTOTYPE(*NAME*, *TYPE*, *FIELD*, *CMP*);

ARB_PROTOTYPE_STATIC(*NAME*, *TYPE*, *FIELD*, *CMP*);

ARB_PROTOTYPE_INSERT(*NAME*, *TYPE*, *ATTR*);

ARB_PROTOTYPE_INSERT_COLOR(*NAME*, *TYPE*, *ATTR*);

ARB_PROTOTYPE_REMOVE(*NAME*, *TYPE*, *ATTR*);

ARB_PROTOTYPE_REMOVE_COLOR(*NAME*, *TYPE*, *ATTR*);

ARB_PROTOTYPE_FIND(*NAME*, *TYPE*, *ATTR*);

ARB_PROTOTYPE_NFIND(*NAME*, *TYPE*, *ATTR*);

ARB_PROTOTYPE_NEXT(*NAME*, *TYPE*, *ATTR*);

ARB_PROTOTYPE_PREV(*NAME*, *TYPE*, *ATTR*);

ARB_PROTOTYPE_MINMAX(*NAME*, *TYPE*, *ATTR*);

ARB_PROTOTYPE_REINSERT(*NAME*, *TYPE*, *ATTR*);

ARB_GENERATE(*NAME*, *TYPE*, *FIELD*, *CMP*);

ARB_GENERATE_STATIC(*NAME*, *TYPE*, *FIELD*, *CMP*);

ARB_GENERATE_INSERT(*NAME*, *TYPE*, *FIELD*, *CMP*, *ATTR*);

ARB_GENERATE_INSERT_COLOR(*NAME*, *TYPE*, *FIELD*, *ATTR*);

ARB_GENERATE_REMOVE(*NAME*, *TYPE*, *FIELD*, *ATTR*);

ARB_GENERATE_REMOVE_COLOR(*NAME*, *TYPE*, *FIELD*, *ATTR*);

ARB_GENERATE_FIND(*NAME*, *TYPE*, *FIELD*, *CMP*, *ATTR*);

ARB_GENERATE_NFIND(*NAME*, *TYPE*, *FIELD*, *CMP*, *ATTR*);

ARB_GENERATE_NEXT(*NAME*, *TYPE*, *FIELD*, *ATTR*);

ARB_GENERATE_PREV(*NAME*, *TYPE*, *FIELD*, *ATTR*);

ARB_GENERATE_MINMAX(*NAME*, *TYPE*, *FIELD*, *ATTR*);

ARB_GENERATE_REINSERT(*NAME*, *TYPE*, *FIELD*, *CMP*, *ATTR*);

ARB<8|16|32>_ENTRY();

ARB<8|16|32>_HEAD(*HEADNAME*, *TYPE*);

size_t
ARB_ALLOCSIZE(*ARB_HEAD *head*, *int<8|16|32>_t maxnodes*, *struct TYPE *elm*);

ARB_INITIALIZER(*ARB_HEAD *head*, *int<8|16|32>_t maxnodes*);

*struct TYPE **
ARB_ROOT(*ARB_HEAD *head*);

bool
ARB_EMPTY(*ARB_HEAD *head*);

bool
ARB_FULL(*ARB_HEAD *head*);

int<8|16|32>_t
ARB_CURNODES(*ARB_HEAD *head*);

int<8|16|32>_t
ARB_MAXNODES(*ARB_HEAD *head*);

*struct TYPE **
ARB_NEXT(*NAME*, *ARB_HEAD *head*, *struct TYPE *elm*);

*struct TYPE **
ARB_PREV(*NAME*, *ARB_HEAD *head*, *struct TYPE *elm*);

*struct TYPE **
ARB_MIN(*NAME*, *ARB_HEAD *head*);

*struct TYPE **
ARB_MAX(*NAME*, *ARB_HEAD *head*);

*struct TYPE **
ARB_FIND(*NAME*, *ARB_HEAD *head*, *struct TYPE *elm*);

*struct TYPE **
ARB_NFIND(*NAME*, *ARB_HEAD *head*, *struct TYPE *elm*);

*struct TYPE **
ARB_LEFT(*struct TYPE *elm*, *ARB_ENTRY NAME*);

int<8|16|32>_t
ARB_LEFTIDX(*struct TYPE *elm*, *ARB_ENTRY NAME*);

*struct TYPE **
ARB_RIGHT(*struct TYPE *elm*, *ARB_ENTRY NAME*);

int<8|16|32>_t

ARB_RIGHTIDX(*struct TYPE *elm*, *ARB_ENTRY NAME*);

*struct TYPE **
ARB_PARENT(*struct TYPE *elm*, *ARB_ENTRY NAME*);

int<8|16|32>_t
ARB_PARENTIDX(*struct TYPE *elm*, *ARB_ENTRY NAME*);

*struct TYPE **
ARB_GETFREE(*ARB_HEAD *head*, *FIELD*);

int<8|16|32>_t
ARB_FREEIDX(*ARB_HEAD *head*);

ARB_FOREACH(*VARNAME*, *NAME*, *ARB_HEAD *head*);

ARB_FOREACH_FROM(*VARNAME*, *NAME*, *POS_VARNAME*);

ARB_FOREACH_SAFE(*VARNAME*, *NAME*, *ARB_HEAD *head*, *TEMP_VARNAME*);

ARB_FOREACH_REVERSE(*VARNAME*, *NAME*, *ARB_HEAD *head*);

ARB_FOREACH_REVERSE_FROM(*VARNAME*, *NAME*, *POS_VARNAME*);

ARB_FOREACH_REVERSE_SAFE(*VARNAME*, *NAME*, *ARB_HEAD *head*, *TEMP_VARNAME*);

void
ARB_INIT(*struct TYPE *elm*, *FIELD*, *ARB_HEAD *head*, *int<8|16|32>_t maxnodes*);

*struct TYPE **
ARB_INSERT(*NAME*, *ARB_HEAD *head*, *struct TYPE *elm*);

*struct TYPE **
ARB_REMOVE(*NAME*, *ARB_HEAD *head*, *struct TYPE *elm*);

*struct TYPE **
ARB_REINSERT(*NAME*, *ARB_HEAD *head*, *struct TYPE *elm*);

void
ARB_RESET_TREE(*ARB_HEAD *head*, *NAME*, *int<8|16|32>_t maxnodes*);

DESCRIPTION

These macros define data structures for and array-based red-black trees. They use a single, continuous chunk of memory, and are useful e.g., when the tree needs to be transferred between userspace and kernel.

In the macro definitions, *TYPE* is the name tag of a user defined structure that must contain a field of type *ARB_ENTRY*, named *ENTRYNAME*. The argument *HEADNAME* is the name tag of a user defined structure that must be declared using the **ARB_HEAD**() macro. The argument *NAME* has to be a unique name prefix for every tree that is defined.

The function prototypes are declared with **ARB_PROTOTYPE**(), or **ARB_PROTOTYPE_STATIC**(). The function bodies are generated with **ARB_GENERATE**(), or **ARB_GENERATE_STATIC**(). See the examples below for further explanation of how these macros are used.

A red-black tree is a binary search tree with the node color as an extra attribute. It fulfills a set of conditions:

1. Every search path from the root to a leaf consists of the same number of black nodes.

2. Each red node (except for the root) has a black parent.

3. Each leaf node is black.

Every operation on a red-black tree is bounded as **O**(*lg n*). The maximum height of a red-black tree is **2lg**(*n + 1*).

ARB_*() trees require entries to be allocated as an array, and uses array indices to link entries together. The maximum number of **ARB_***() tree entries is therefore constrained by the minimum of array size and choice of signed integer data type used to store array indices. Use **ARB_ALLOCSIZE**() to compute the size of memory chunk to allocate.

A red-black tree is headed by a structure defined by the **ARB_HEAD**() macro. A structure is declared with either of the following:

ARB<8|16|32>_HEAD(*HEADNAME, TYPE*) *head*;

where *HEADNAME* is the name of the structure to be defined, and struct *TYPE* is the type of the elements to be inserted into the tree.

The **ARB_HEAD**() variant includes a suffix denoting the signed integer data type size (in bits) used to store array indices. For example, **ARB_HEAD8**() creates a red-black tree head strucutre with 8-bit signed array indices capable of indexing up to 128 entries.

The **ARB_ENTRY**() macro declares a structure that allows elements to be connected in the tree. Similarly to the **ARB<8|16|32>_HEAD**() macro, the **ARB_ENTRY**() variant includes a suffix denoting the signed integer data type size (in bits) used to store array indices. Entries should use the same number of bits as the tree head structure they will be linked into.

In order to use the functions that manipulate the tree structure, their prototypes need to be declared with the **ARB_PROTOTYPE**() or **ARB_PROTOTYPE_STATIC**() macro, where *NAME* is a unique identifier for this particular tree. The *TYPE* argument is the type of the structure that is being managed by the tree. The *FIELD* argument is the name of the element defined by **ARB_ENTRY**(). Individual prototypes can be declared with **ARB_PROTOTYPE_INSERT**(), **ARB_PROTOTYPE_INSERT_COLOR**(), **ARB_PROTOTYPE_REMOVE**(), **ARB_PROTOTYPE_REMOVE_COLOR**(), **ARB_PROTOTYPE_FIND**(), **ARB_PROTOTYPE_NFIND**(), **ARB_PROTOTYPE_NEXT**(), **ARB_PROTOTYPE_PREV**(), **ARB_PROTOTYPE_MINMAX**(), and **ARB_PROTOTYPE_REINSERT**() in case not all functions are required. The individual prototype macros expect *NAME*, *TYPE*, and *ATTR* arguments. The *ATTR* argument must be empty for global functions or *static* for static functions.

The function bodies are generated with the **ARB_GENERATE**() or **ARB_GENERATE_STATIC**() macro. These macros take the same arguments as the **ARB_PROTOTYPE**() and **ARB_PROTOTYPE_STATIC**() macros, but should be used only once. As an alternative individual function bodies are generated with the **ARB_GENERATE_INSERT**(), **ARB_GENERATE_INSERT_COLOR**(), **ARB_GENERATE_REMOVE**(), **ARB_GENERATE_REMOVE_COLOR**(), **ARB_GENERATE_FIND**(), **ARB_GENERATE_NFIND**(), **ARB_GENERATE_NEXT**(), **ARB_GENERATE_PREV**(), **ARB_GENERATE_MINMAX**(), and **ARB_GENERATE_REINSERT**() macros.

Finally, the *CMP* argument is the name of a function used to compare tree nodes with each other. The function takes two arguments of type *struct TYPE **. If the first argument is smaller than the second, the function returns a value smaller than zero. If they are equal, the function returns zero. Otherwise, it should return a value greater than zero. The compare function defines the order of the tree elements.

The **ARB_INIT**() macro initializes the tree referenced by *head*, with the array length of *maxnodes*.

The red-black tree can also be initialized statically by using the **ARB_INITIALIZER**() macro:

ARB<8|16|32>_HEAD(*HEADNAME*, *TYPE*) *head* = **ARB_INITIALIZER**(*&head*, *maxnodes*);

The **ARB_INSERT**() macro inserts the new element *elm* into the tree.

The **ARB_REMOVE**() macro removes the element *elm* from the tree pointed by *head*.

The **ARB_FIND**() and **ARB_NFIND**() macros can be used to find a particular element in the tree.

```
struct TYPE find, *res;
find.key = 30;
res = ARB_FIND(NAME, head, &find);
```

The **ARB_ROOT**(), **ARB_MIN**(), **ARB_MAX**(), **ARB_NEXT**(), and **ARB_PREV**() macros can be used to traverse the tree:

```
for (np = ARB_MIN(NAME, &head); np != NULL; np = ARB_NEXT(NAME, &head, np))
```

Or, for simplicity, one can use the **ARB_FOREACH**() or **ARB_FOREACH_REVERSE**() macro:

ARB_FOREACH(*np*, *NAME*, *head*)

The macros **ARB_FOREACH_SAFE**() and **ARB_FOREACH_REVERSE_SAFE**() traverse the tree referenced by head in a forward or reverse direction respectively, assigning each element in turn to np. However, unlike their unsafe counterparts, they permit both the removal of np as well as freeing it from within the loop safely without interfering with the traversal.

Both **ARB_FOREACH_FROM**() and **ARB_FOREACH_REVERSE_FROM**() may be used to continue an interrupted traversal in a forward or reverse direction respectively. The head pointer is not required. The pointer to the node from where to resume the traversal should be passed as their last argument, and will be overwritten to provide safe traversal.

The **ARB_EMPTY**() macro should be used to check whether a red-black tree is empty.

Given that ARB trees have an intrinsic upper bound on the number of entries, some ARB-specific additional macros are defined. The **ARB_FULL**() macro returns a boolean indicating whether the current number of tree entries equals the tree's maximum. The **ARB_CURNODES**() and **ARB_MAXNODES**() macros return the current and maximum number of entries respectively. The **ARB_GETFREE**() macro returns a pointer to the next free entry in the array of entries, ready to be linked into the tree. The **ARB_INSERT**() returns NULL if the element was inserted in the tree successfully, otherwise they return a pointer to the element with the colliding key.

Accordingly, **ARB_REMOVE**() returns the pointer to the removed element otherwise they return NULL to indicate an error.

The **ARB_REINSERT**() macro updates the position of the element *elm* in the tree. This must be called if a member of a **tree** is modified in a way that affects comparison, such as by modifying a node's key. This is a lower overhead alternative to removing the element and reinserting it again.

The **ARB_RESET_TREE**() macro discards the tree topology. It does not modify embedded object values or the free list.

SEE ALSO

queue(3), tree(3)

HISTORY

The **ARB** macros first appeared in FreeBSD 13.0.

AUTHORS

The **ARB** macros were implemented by Lawrence Stewart <*lstewart@FreeBSD.org*>, based on tree(3) macros written by
Niels Provos.