NAME

CIRCLEQ_EMPTY, CIRCLEQ_ENTRY, CIRCLEQ_FIRST, CIRCLEQ_FOREACH, CIRCLEQ_FOREACH_REVERSE, CIRCLEQ_HEAD, CIRCLEQ_HEAD_INITIALIZER, CIRCLEQ_INIT, CIRCLEQ_INSERT_AFTER, CIRCLEQ_INSERT_BEFORE, CIRCLEQ_INSERT_HEAD, CIRCLEQ_INSERT_TAIL, CIRCLEQ_LAST, CIRCLEQ_LOOP_NEXT, CIRCLEQ_LOOP_PREV, CIRCLEQ_NEXT, CIRCLEQ_PREV, CIRCLEQ_REMOVE − implementation of a doubly linked circular queue

LIBRARY

Standard C library (*libc*, *−lc*)

SYNOPSIS

#include <sys/queue.h>

CIRCLEQ_ENTRY(TYPE);

CIRCLEQ_HEAD(HEADNAME, TYPE);
CIRCLEQ_HEAD CIRCLEQ_HEAD_INITIALIZER(CIRCLEQ_HEAD *head***);**
void CIRCLEQ_INIT(CIRCLEQ_HEAD *****head*);**

int CIRCLEQ_EMPTY(CIRCLEQ_HEAD *****head*);**

void CIRCLEQ_INSERT_HEAD(CIRCLEQ_HEAD *****head*,**
struct TYPE *****elm***, CIRCLEQ_ENTRY** *NAME***);**
void CIRCLEQ_INSERT_TAIL(CIRCLEQ_HEAD *****head*,**
struct TYPE *****elm***, CIRCLEQ_ENTRY** *NAME***);**
void CIRCLEQ_INSERT_BEFORE(CIRCLEQ_HEAD *****head***, struct TYPE *****listelm*****,**
struct TYPE *****elm***, CIRCLEQ_ENTRY** *NAME***);**
void CIRCLEQ_INSERT_AFTER(CIRCLEQ_HEAD *****head***, struct TYPE *****listelm*****,**
struct TYPE *****elm***, CIRCLEQ_ENTRY** *NAME***);**

struct TYPE *CIRCLEQ_FIRST(CIRCLEQ_HEAD *****head*);**
struct TYPE *CIRCLEQ_LAST(CIRCLEQ_HEAD *****head*);**
struct TYPE *CIRCLEQ_PREV(struct TYPE *****elm***, CIRCLEQ_ENTRY** *NAME***);**
struct TYPE *CIRCLEQ_NEXT(struct TYPE *****elm***, CIRCLEQ_ENTRY** *NAME***);**
struct TYPE *CIRCLEQ_LOOP_PREV(CIRCLEQ_HEAD *****head*,**
struct TYPE *****elm***, CIRCLEQ_ENTRY** *NAME***);**
struct TYPE *CIRCLEQ_LOOP_NEXT(CIRCLEQ_HEAD *****head*,**
struct TYPE *****elm***, CIRCLEQ_ENTRY** *NAME***);**

CIRCLEQ_FOREACH(struct TYPE *****var***, CIRCLEQ_HEAD *****head***,**
CIRCLEQ_ENTRY *NAME***);**
CIRCLEQ_FOREACH_REVERSE(struct TYPE *****var***, CIRCLEQ_HEAD *****head***,**
CIRCLEQ_ENTRY *NAME***);**

void CIRCLEQ_REMOVE(CIRCLEQ_HEAD *****head***, struct TYPE *****elm*****,**
CIRCLEQ_ENTRY *NAME***);**

DESCRIPTION

These macros define and operate on doubly linked circular queues.

In the macro definitions,*TYPE* is the name of a user -defined structure, that must contain a field of type *CIRCLEQ_ENTRY*, named *NAME*. The argument *HEADNAME* is the name of a user-defined structure that must be declared using the macro **CIRCLEQ_HEAD**().

Creation

A circular queue is headed by a structure defined by the **CIRCLEQ_HEAD**() macro. This structure contains a pair of pointers, one to the first element in the queue and the other to the last element in the queue. The elements are doubly linked so that an arbitrary element can be removed without traversing the queue. New elements can be added to the queue after an existing element, before an existing element, at the head of the queue, or at the end of the queue. A*CIRCLEQ_HEAD* structure is declared as follo ws:

```
CIRCLEQ_HEAD(HEADNAME, TYPE) head;
```

where *struct HEADNAME* is the structure to be defined, and *struct TYPE* is the type of the elements to be linked into the queue. A pointer to the head of the queue can later be declared as:

```
struct HEADNAME *headp;
```

(The names *head* and *headp* are user selectable.)

CIRCLEQ_ENTRY() declares a structure that connects the elements in the queue.

CIRCLEQ_HEAD_INITIALIZER() evaluates to an initializer for the queue *head*.

CIRCLEQ_INIT() initializes the queue referenced by *head*.

CIRCLEQ_EMPTY() evaluates to true if there are no items on the queue.

Insertion

CIRCLEQ_INSERT_HEAD() inserts the new element *elm* at the head of the queue.

CIRCLEQ_INSERT_TAIL() inserts the new element *elm* at the end of the queue.

CIRCLEQ_INSERT_BEFORE() inserts the new element *elm* before the element *listelm*.

CIRCLEQ_INSERT_AFTER() inserts the new element *elm* after the element *listelm*.

Traversal

CIRCLEQ_FIRST() returns the first item on the queue.

CIRCLEQ_LAST() returns the last item on the queue.

CIRCLEQ_PREV() returns the previous item on the queue, or*&head* if this item is the first one.

CIRCLEQ_NEXT() returns the next item on the queue, or*&head* if this item is the last one.

CIRCLEQ_LOOP_PREV() returns the previous item on the queue. If *elm* is the first element on the queue, the last element is returned.

CIRCLEQ_LOOP_NEXT() returns the next item on the queue. If *elm* is the last element on the queue, the first element is returned.

CIRCLEQ_FOREACH() traverses the queue referenced by *head* in the forward direction, assigning each element in turn to *var*. *var* is set to *&head* if the loop completes normally, or if there were no elements.

CIRCLEQ_FOREACH_REVERSE() traverses the queue referenced by *head* in the reverse direction, assigning each element in turn to *var*.

Removal

CIRCLEQ_REMOVE() removes the element *elm* from the queue.

RETURN VALUE

CIRCLEQ_EMPTY() returns nonzero if the queue is empty, and zero if the queue contains at least one entry.

CIRCLEQ_FIRST(), **CIRCLEQ_LAST**(), **CIRCLEQ_LOOP_PREV**(), and **CIRCLEQ_LOOP_NEXT**() return a pointer to the first, last, previous, or next *TYPE* structure, respectively.

CIRCLEQ_PREV(), and **CIRCLEQ_NEXT**() are similar to their **CIRCLEQ_LOOP_***() counterparts, except that if the argument is the first or last element, respectively, they return*&head*.

CIRCLEQ_HEAD_INITIALIZER() returns an initializer that can be assigned to the queue *head*.

STANDARDS

Not in POSIX.1, POSIX.1-2001, or POSIX.1-2008. Present on the BSDs (CIRCLEQ macros first appeared in 4.4BSD).

BUGS

CIRCLEQ_FOREACH() and **CIRCLEQ_FOREACH_REVERSE**() don't allow *var* to be removed or freed within the loop, as it would interfere with the traversal. **CIRCLEQ_FOREACH_SAFE**() and **CIRCLEQ_FOREACH_REVERSE_SAFE**(), which are present on the BSDs but are not present in glibc, fix this limitation by allowing *var* to safely be removed from the list and freed from within the loop without

interfering with the traversal.

EXAMPLES

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/queue.h>

struct entry {
    int data;
    CIRCLEQ_ENTRY(entry) entries;          /* Queue */
};

CIRCLEQ_HEAD(circlehead, entry);

int
main(void)
{
    struct entry *n1, *n2, *n3, *np;
    struct circlehead head;                /* Queue head */
    int i;

    CIRCLEQ_INIT(&head);                   /* Initialize the queue */

    n1 = malloc(sizeof(struct entry));     /* Insert at the head */
    CIRCLEQ_INSERT_HEAD(&head, n1, entries);

    n1 = malloc(sizeof(struct entry));     /* Insert at the tail */
    CIRCLEQ_INSERT_TAIL(&head, n1, entries);

    n2 = malloc(sizeof(struct entry));     /* Insert after */
    CIRCLEQ_INSERT_AFTER(&head, n1, n2, entries);

    n3 = malloc(sizeof(struct entry));     /* Insert before */
    CIRCLEQ_INSERT_BEFORE(&head, n2, n3, entries);

    CIRCLEQ_REMOVE(&head, n2, entries);    /* Deletion */
    free(n2);
                                           /* Forward traversal */
    i = 0;
    CIRCLEQ_FOREACH(np, &head, entries)
        np->data = i++;
                                           /* Reverse traversal */
    CIRCLEQ_FOREACH_REVERSE(np, &head, entries)
        printf("%i\n", np->data);
                                           /* Queue deletion */
    n1 = CIRCLEQ_FIRST(&head);
    while (n1 != (void *)&head) {
        n2 = CIRCLEQ_NEXT(n1, entries);
        free(n1);
        n1 = n2;
    }
    CIRCLEQ_INIT(&head);

    exit(EXIT_SUCCESS);
```

```
}
```

SEE ALSO

insque(3), **queue**(7)