



*Full credit is given to the above companies including the OS that this PDF file was generated!*

## **Red Hat Enterprise Linux Release 9.2 Manual Pages on 'SLIST\_INSERT\_AFTER.3' command**

**\$ man SLIST\_INSERT\_AFTER.3**

SLIST(3)            Linux Programmer's Manual            SLIST(3)

### NAME

SLIST\_EMPTY, SLIST\_ENTRY, SLIST\_FIRST, SLIST\_FOREACH, SLIST\_HEAD, SLIST\_HEAD\_INITIALIZER, SLIST\_INIT, SLIST\_INSERT\_AFTER, SLIST\_INSERT\_HEAD, SLIST\_NEXT, SLIST\_REMOVE, SLIST\_REMOVE\_HEAD - implementation of a singly linked list

### SYNOPSIS

```
#include <sys/queue.h>

int SLIST_EMPTY(SLIST_HEAD *head);

SLIST_ENTRY(TYPE);

struct TYPE *SLIST_FIRST(SLIST_HEAD *head);

SLIST_FOREACH(struct TYPE *var, SLIST_HEAD *head, SLIST_ENTRY NAME);

SLIST_HEAD(HEADNAME, TYPE);

SLIST_HEAD SLIST_HEAD_INITIALIZER(SLIST_HEAD head);

void SLIST_INIT(SLIST_HEAD *head);

void SLIST_INSERT_AFTER(struct TYPE *listelm, struct TYPE *elm,
                        SLIST_ENTRY NAME);

void SLIST_INSERT_HEAD(SLIST_HEAD *head, struct TYPE *elm,
                       SLIST_ENTRY NAME);

struct TYPE *SLIST_NEXT(struct TYPE *elm, SLIST_ENTRY NAME);

void SLIST_REMOVE(SLIST_HEAD *head, struct TYPE *elm, SLIST_ENTRY NAME);

void SLIST_REMOVE_HEAD(SLIST_HEAD *head, SLIST_ENTRY NAME);
```

### DESCRIPTION

These macros define and operate on doubly linked lists.

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

A singly linked list is headed by a structure defined by the `SLIST_HEAD()` macro. This structure contains a single pointer to the first element on the list. The elements are singly linked for minimum space and pointer manipulation overhead at the expense of  $O(n)$  removal for arbitrary elements. New elements can be added to the list after an existing element or at the head of the list. An `SLIST_HEAD` structure is declared as follows:

```
SLIST_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 list. A pointer to the head of the list can later be declared as:

```
struct HEADNAME *headp;
```

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

The macro `SLIST_HEAD_INITIALIZER()` evaluates to an initializer for the list head.

The macro `SLIST_EMPTY()` evaluates to true if there are no elements in the list.

The macro `SLIST_ENTRY()` declares a structure that connects the elements in the list.

The macro `SLIST_FIRST()` returns the first element in the list or `NULL` if the list is empty.

The macro `SLIST_FOREACH()` traverses the list referenced by `head` in the forward direction, assigning each element in turn to `var`.

The macro `SLIST_INIT()` initializes the list referenced by `head`.

The macro `SLIST_INSERT_HEAD()` inserts the new element `elm` at the head of the list.

The macro `SLIST_INSERT_AFTER()` inserts the new element `elm` after the element `listelm`.

The macro `SLIST_NEXT()` returns the next element in the list.

The macro `SLIST_REMOVE_HEAD()` removes the element `elm` from the head of the list. For optimum efficiency, elements being removed from the head of the list should explicitly use this macro instead of the generic `SLIST_REMOVE` macro.

The macro `SLIST_REMOVE()` removes the element `elm` from the list.

## RETURN VALUE

`SLIST_EMPTY()` returns nonzero if the list is empty, and zero if the list contains at least one entry.

`SLIST_FIRST()`, and `SLIST_NEXT()` return a pointer to the first or next `TYPE` structure, respectively.

`SLIST_HEAD_INITIALIZER()` returns an initializer that can be assigned to the list head.

## CONFORMING TO

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

## BUGS

The macro `SLIST_FOREACH()` doesn't allow `var` to be removed or freed within the loop, as it would interfere with the traversal. The macro `SLIST_FOREACH_SAFE()`, which is present on the BSDs but is not present in glibc, fixes 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;
    SLIST_ENTRY(entry) entries;    /* Singly linked List. */
};

SLIST_HEAD(slisthead, entry);
```

```

int
main(void)
{
    struct entry *n1, *n2, *n3, *np;
    struct slisthead head;          /* Singly linked List
                                     head. */
    SLIST_INIT(&head);             /* Initialize the queue. */
    n1 = malloc(sizeof(struct entry)); /* Insert at the head. */
    SLIST_INSERT_HEAD(&head, n1, entries);
    n2 = malloc(sizeof(struct entry)); /* Insert after. */
    SLIST_INSERT_AFTER(n1, n2, entries);
    SLIST_REMOVE(&head, n2, entry, entries); /* Deletion. */
    free(n2);
    n3 = SLIST_FIRST(&head);
    SLIST_REMOVE_HEAD(&head, entries); /* Deletion from the head. */
    free(n3);
    for (int i = 0; i < 5; i++) {
        n1 = malloc(sizeof(struct entry));
        SLIST_INSERT_HEAD(&head, n1, entries);
        n1->data = i;
    }
                                     /* Forward traversal. */
    SLIST_FOREACH(np, &head, entries)
        printf("%i\n", np->data);
    while (!SLIST_EMPTY(&head)) { /* List Deletion. */
        n1 = SLIST_FIRST(&head);
        SLIST_REMOVE_HEAD(&head, entries);
        free(n1);
    }
    SLIST_INIT(&head);
    exit(EXIT_SUCCESS);
}

```

insque(3), queue(7)

## COLOPHON

This page is part of release 5.10 of the Linux man-pages project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at <https://www.kernel.org/doc/man-pages/>.

GNU

2020-10-21

SLIST(3)