NAME

hsearch, hcreate, hdestroy – manage hash search tables

SYNOPSIS

#include <search.h>

ENTRY *hsearch(ENTRY *item***, ACTION** *action***);**

int hcreate (size_t *mekments***);**

void hdestroy(void);

MT-LEVEL

Safe

DESCRIPTION

hsearch( ) is a hash-table search routine generalized from Knuth (6.4) Algorithm D. It returns a pointer into a hash table indicating the location at which an entry can be found. The comparison function used by **hsearch( )** is **strcmp( )** (see **string**(3C)). *item* is a structure of type **ENTRY** (defined in the **<search.h>** header) containing two pointers: *item.key* points to the comparison key, and *item.data* points to any other data to be associated with that key. (Pointers to types other than void should be cast to pointer-to-void.) *action* is a member of an enumeration type **ACTION** (defined in **<search.h>**) indicating the disposition of the entry if it cannot be found in the table. **ENTER** indicates that the item should be inserted in the table at an appropriate point. Given a duplicate of an existing item, the new item is not entered and **hsearch( )** returns a pointer to the existing item. **FIND** indicates that no entry should be made. Unsuccessful resolution is indicated by the return of a null pointer.

hcreate( ) allocates sufficient space for the table, and must be called before **hsearch( )** is used. *nel* is an estimate of the maximum number of entries that the table will contain. This number may be adjusted upward by the algorithm in order to obtain certain mathematically favorable circumstances.

hdestroy( ) destroys the search table, and may be followed by another call to **hcreate( )**.

RETURN VALUES

hsearch( ) returns a null pointer if either the action is **FIND** and the item could not be found or the action is **ENTER** and the table is full.

hcreate( ) returns zero if it cannot allocate sufficient space for the table.

EXAMPLE

The following example will read in strings followed by two numbers and store them in a hash table, discarding duplicates. It will then read in strings and find the matching entry in the hash table and print it out.

```
#include <stdio.h>
#include <search.h>
#include <string.h>
#include <stdlib.h>

struct info {                   /* this is the info stored in table */
        int age, room;          /* other than the key */
};
```

```
#define NUM_EMPL     5000     /* # of elements in search table */

main( )
{
                    /* space to store strings */
        char string_space[NUM_EMPL*20];
                    /* space to store employee info */
        struct info info_space[NUM_EMPL];
                    /* next avail space in string_space */
        char *str_ptr = string_space;
                    /* next avail space in info_space */
        struct info *info_ptr = info_space;
        ENTRY item, *found_item;
                    /* name to look for in table */
        char name_to_find[30];
        int i = 0;

                    /* create table */
        (void) hcreate(NUM_EMPL);
        while (scanf("%s%d%d", str_ptr, &info_ptr->age,
              &info_ptr->room) != EOF && i++ < NUM_EMPL) {
                    /* put info in structure, and structure in item */
                item.key = str_ptr;
                item.data = (void *)info_ptr;
                str_ptr += strlen(str_ptr) + 1;
                info_ptr++;
                    /* put item into table */
                (void) hsearch(item, ENTER);
        }

                    /* access table */
        item.key = name_to_find;
        while (scanf("%s", item.key) != EOF) {
            if ((found_item = hsearch(item, FIND)) != NULL) {
                    /* if item is in the table */
                (void)printf("found %s, age = %d, room = %d\n",
                        found_item->key,
                        ((struct info *)found_item->data)->age,
                        ((struct info *)found_item->data)->room);
            } else {
                (void)printf("no such employee %s\n",
                        name_to_find)
            }
        }
        return 0;
```

}

SEE ALSO

bsearch(3C), **lsearch**(3C), **malloc**(3C), **string**(3C), **tsearch**(3C), **malloc**(3X)

The Art of Computer Programming, Volume 3, Sorting and Searching by Donald E. Knuth, published by Addison-Wesley Publishing Company, 1973.

NOTES

hsearch( ) and **hcreate( )** use **malloc**(3C) to allocate space.

Only one hash search table may be active at any given time.