NAME

bsearch – binary search a sorted table

SYNOPSIS

#include <stdlib.h>

void *bsearch(const void ***key*, const void ***base***, size_t** *nel***, size_t** *size***,
int (***compar***)(const void *, const void *));**

MT-LEVEL

Safe

DESCRIPTION

bsearch( ) is a binary search routine generalized from Knuth (6.2.1) Algorithm B. It returns a pointer into a table (an array) indicating where a datum may be found or a null pointer if the datum cannot be found. The table must be previously sorted in increasing order according to a comparison function pointed to by *compar*. *key* points to a datum instance to be sought in the table. *base* points to the element at the base of the table. *nel* is the number of elements in the table. *size* is the number of bytes in each element. The function pointed to by *compar* is called with two arguments that point to the elements being compared. The function must return an integer less than, equal to, or greater than 0 as accordingly the first argument is to be considered less than, equal to, or greater than the second.

RETURN VALUES

A null pointer is returned if the key cannot be found in the table.

EXAMPLES

The example below searches a table containing pointers to nodes consisting of a string and its length. The table is ordered alphabetically on the string in the node pointed to by each entry.

This program reads in strings and either finds the corresponding node and prints out the string and its length, or prints an error message.

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

struct node {                   /* these are stored in the table */
        char *string;
        int length;
};
static struct node table[] = {  /* table to be searched */
        { "asparagus", 10 },
        { "beans", 6 },
        { "tomato", 7 },
        { "watermelon", 11 },
};

main( )
{
        struct node *node_ptr, node;
```

```
        /* routine to compare 2 nodes */
        static int node_compare(const void *, const void *);
        char str_space[20];   /* space to read string into */

        node.string = str_space;
        while (scanf("%20s", node.string) != EOF) {
                node_ptr = bsearch( &node,
                           table, sizeof(table)/sizeof(struct node),
                           sizeof(struct node), node_compare);
                if (node_ptr != NULL) {
                        (void) printf("string = %20s, length = %d\n",
                                node_ptr->string, node_ptr->length);
                } else {
                        (void)printf("not found: %20s\n", node.string);
                }
        }
        return(0);
}

/* routine to compare two nodes based on an  */
/* alphabetical ordering of the string field */
static int
node_compare(const void *node1, const void *node2) {
        return (strcmp(
                        ((const struct node *)node1)->string,
                        ((const struct node *)node2)->string));
}
```

SEE ALSO

hsearch(3C), **lsearch**(3C), **qsort**(3C), **tsearch**(3C)

NOTES

The pointers to the key and the element at the base of the table should be of type pointer-to-*element*.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

If the number of elements in the table is less than the size reserved for the table, *nel* should be the lower number.