std::bsearch

Defined in header <cstdlib>

```
void* bsearch( const void* key, const void* ptr, std::size_t count,
               std::size_t size, /*compare-pred*/* comp );
void* bsearch( const void* key, const void* ptr, std::size_t count,
               std::size_t size, /*c-compare-pred*/* comp );                    (1)
```

```
extern "C++" using /*compare-pred*/ = int(const void*, const void*); // exposition-only
extern "C" using /*c-compare-pred*/ = int(const void*, const void*); // exposition-only   (2)
```

Finds an element equal to element pointed to by key in an array pointed to by ptr. The array contains count elements of size bytes each and must be partitioned with respect to the object pointed to by key, that is, all the elements that compare less than must appear before all the elements that compare equal to, and those must appear before all the elements that compare greater than the key object. A fully sorted array satisfies these requirements. The elements are compared using function pointed to by comp.

The behavior is undefined if the array is not already partitioned in ascending order with respect to key, according to the same criterion that comp uses.

If the array contains several elements that comp would indicate as equal to the element searched for, then it is unspecified which element the function will return as the result.

Parameters

key	-	pointer to the element to search for
ptr	-	pointer to the array to examine
count	-	number of element in the array
size	-	size of each element in the array in bytes
comp	-	comparison function which returns a negative integer value if the first argument is *less* than the second, a positive integer value if the first argument is *greater* than the second and zero if the arguments are equivalent. key is passed as the first argument, an element from the array as the second. The signature of the comparison function should be equivalent to the following: `int cmp(const void *a, const void *b);` The function must not modify the objects passed to it and must return consistent results when called for the same objects, regardless of their positions in the array.

Return value

Pointer to the found element or null pointer if the element has not been found.

Notes

Despite the name, neither C nor POSIX standards require this function to be implemented using binary search or make any complexity guarantees.

The two overloads provided by the C++ standard library are distinct because the types of the parameter comp are distinct (language linkage is part of its type).

Example

Run this code

```
#include <iostream>
#include <cstdlib>
#include <array>

template <typename T>
int compare(const void *a, const void *b) {
    const auto &arg1 = *(static_cast<const T*>(a));
    const auto &arg2 = *(static_cast<const T*>(b));
    const auto cmp = arg1 <=> arg2;
    return cmp < 0 ? -1
        :  cmp > 0 ? +1
        :  0;
}

int main() {
    std::array arr { 1, 2, 3, 4, 5, 6, 7, 8 };

    for (const int key : { 4, 8, 9 }) {

        const int* p = static_cast<int*>(
            std::bsearch(&key,
                arr.data(),
                arr.size(),
                sizeof(decltype(arr)::value_type),
                compare<int>));

        std::cout << "value " << key;
        (p) ? std::cout << " found at position " << (p - arr.data()) << '\n'
            : std::cout << " not found\n";
    }
}
```

Output:

```
value 4 found at position 3
value 8 found at position 7
value 9 not found
```

See also

qsort	sorts a range of elements with unspecified type (function)
equal_range	returns range of elements matching a specific key (function template)

C documentation for **bsearch**

Retrieved from "https://en.cppreference.com/mwiki/index.php?title=cpp/algorithm/bsearch&oldid=129860"