qsort, qsort_s

Defined in header <stdlib.h>

```
void qsort( void *ptr, size_t count, size_t size,
            int (*comp)(const void *, const void *) );                    (1)
errno_t qsort_s( void *ptr, rsize_t count, rsize_t size,
                 int (*comp)(const void *, const void *, void *),         (2)   (since C11)
                 void *context );
```

1) Sorts the given array pointed to by `ptr` in ascending order. The array contains `count` elements of `size` bytes. Function pointed to by `comp` is used for object comparison.

2) Same as (1), except that the additional context parameter `context` is passed to `comp` and that the following errors are detected at runtime and call the currently installed constraint handler function:

- `count` or `size` is greater than `RSIZE_MAX`
- `ptr` or `comp` is a null pointer (unless `count` is zero)

As with all bounds-checked functions, `qsort_s` is only guaranteed to be available if `__STDC_LIB_EXT1__` is defined by the implementation and if the user defines `__STDC_WANT_LIB_EXT1__` to the integer constant 1 before including `stdlib.h`.

If `comp` indicates two elements as equivalent, their order in the resulting sorted array is unspecified.

Parameters

ptr - pointer to the array to sort

count - number of elements 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.

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.

context - additional information (e.g., collating sequence), passed to `comp` as the third argument

Return value

1) (none)

2) zero on success, non-zero if a runtime constraints violation was detected

Notes

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

Unlike other bounds-checked functions, `qsort_s` does not treat arrays of zero size as a runtime constraint violation and instead returns successfully without altering the array (the other function that accepts arrays of zero size is `bsearch_s`).

Until `qsort_s`, users of `qsort` often used global variables to pass additional context to the comparison function.

Example

Run this code

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

```
int compare_ints(const void* a, const void* b)
{
    int arg1 = *(const int*)a;
    int arg2 = *(const int*)b;

    if (arg1 < arg2) return -1;
    if (arg1 > arg2) return 1;
    return 0;

    // return (arg1 > arg2) - (arg1 < arg2); // possible shortcut
    // return arg1 - arg2; // erroneous shortcut (fails if INT_MIN is present)
}

int main(void)
{
    int ints[] = { -2, 99, 0, -743, 2, INT_MIN, 4 };
    int size = sizeof ints / sizeof *ints;

    qsort(ints, size, sizeof(int), compare_ints);

    for (int i = 0; i < size; i++) {
        printf("%d ", ints[i]);
    }

    printf("\n");
}
```

Output:

```
-2147483648 -743 -2 0 2 4 99
```

References

- C11 standard (ISO/IEC 9899:2011):
  - 7.22.5.2 The qsort function (p: 355-356)
  - K.3.6.3.2 The qsort_s function (p: 609)
- C99 standard (ISO/IEC 9899:1999):
  - 7.20.5.2 The qsort function (p: 319)
- C89/C90 standard (ISO/IEC 9899:1990):
  - 4.10.5.2 The qsort function

See also

bsearch **bsearch_s** (C11)	searches an array for an element of unspecified type (function)

C++ documentation for **qsort**