realloc

Defined in header <stdlib.h>

```
void *realloc( void *ptr, size_t new_size );
```

Reallocates the given area of memory. It must be previously allocated by malloc(), calloc() or realloc() and not yet freed with a call to free or realloc. Otherwise, the results are undefined.

The reallocation is done by either:

a) expanding or contracting the existing area pointed to by ptr, if possible. The contents of the area remain unchanged up to the lesser of the new and old sizes. If the area is expanded, the contents of the new part of the array are undefined.

b) allocating a new memory block of size new_size bytes, copying memory area with size equal the lesser of the new and the old sizes, and freeing the old block.

If there is not enough memory, the old memory block is not freed and null pointer is returned.

If ptr is NULL, the behavior is the same as calling `malloc(new_size)`.

Otherwise,

if new_size is zero, the behavior is implementation defined (null pointer may be returned (in which case the old memory block may or may not be freed), or some non-null pointer may be returned that may not be used to access storage). Such usage is deprecated (via DR 400 (http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2396.htm#dr_400)). (since C17) (until C23)
if new_size is zero, the behavior is undefined. (since C23)
realloc is thread-safe: it behaves as though only accessing the memory locations visible through its argument, and not any static storage. A previous call to free or realloc that deallocates a region of memory *synchronizes-with* a call to any allocation function, including realloc that allocates the same or a part of the same region of memory. This synchronization occurs after any access to the memory by the deallocating function and before any access to the memory by realloc. There is a single total order of all allocation and deallocation functions operating on each particular region of memory. (since C11)

Parameters

ptr	-	pointer to the memory area to be reallocated
new_size	-	new size of the array in bytes

Return value

On success, returns the pointer to the beginning of newly allocated memory. To avoid a memory leak, the returned pointer must be deallocated with free() or realloc(). The original pointer ptr is invalidated and any access to it is undefined behavior (even if reallocation was in-place).

On failure, returns a null pointer. The original pointer ptr remains valid and may need to be deallocated with free() or realloc().

Notes

Originally (in C89), support for zero size was added to accommodate code such as

```
OBJ *p = calloc(0, sizeof(OBJ)); // "zero-length" placeholder
...
while(1) {
    p = realloc(p, c * sizeof(OBJ)); // reallocations until size settles
    ... // code that may change c or break out of loop
}
```

Example

Run this code

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

void print_storage_info(const int* next, const int* prev, int ints) {
    if (next) {
        printf("%s location: %p. Size: %d ints (%ld bytes).\n",
               (next != prev ? "New" : "Old"), (void*)next, ints, ints * sizeof(int));
    } else {
        printf("Allocation failed.\n");
    }
}

int main(void)
{
    const int pattern[] = {1, 2, 3, 4, 5, 6, 7, 8};
    const int pattern_size = sizeof pattern / sizeof(int);
    int *next = NULL, *prev = NULL;

    if ((next = (int*)malloc(pattern_size * sizeof *next))) { // allocates an array
        memcpy(next, pattern, sizeof pattern); // fills the array
        print_storage_info(next, prev, pattern_size);
    } else {
        return EXIT_FAILURE;
    }

    // Reallocate in cycle using the following values as a new storage size.
    const int realloc_size[] = {10, 12, 512, 32768, 65536, 32768};

    for (int i = 0; i != sizeof realloc_size / sizeof(int); ++i) {
        if ((next = (int*)realloc(prev = next, realloc_size[i] * sizeof(int)))) {
            print_storage_info(next, prev, realloc_size[i]);
            assert(!memcmp(next, pattern, sizeof pattern));  // is pattern held
        } else {  // if realloc failed, the original pointer needs to be freed
            free(prev);
            return EXIT_FAILURE;
        }
    }

    free(next); // finally, frees the storage
    return EXIT_SUCCESS;
}
```

Possible output:

```
New location: 0x144c010. Size: 8 ints (32 bytes).
Old location: 0x144c010. Size: 10 ints (40 bytes).
New location: 0x144c450. Size: 12 ints (48 bytes).
Old location: 0x144c450. Size: 512 ints (2048 bytes).
Old location: 0x144c450. Size: 32768 ints (131072 bytes).
New location: 0x7f490c5bd010. Size: 65536 ints (262144 bytes).
Old location: 0x7f490c5bd010. Size: 32768 ints (131072 bytes).
```

References

- C17 standard (ISO/IEC 9899:2018):
  - 7.22.3.5 The realloc function (p: 254)
- C11 standard (ISO/IEC 9899:2011):
  - 7.22.3.5 The realloc function (p: 349)
- C99 standard (ISO/IEC 9899:1999):
  - 7.20.3.4 The realloc function (p: 314)
- C89/C90 standard (ISO/IEC 9899:1990):
  - 4.10.3.4 The realloc function

See also

C++ documentation for **realloc**

Retrieved from "https://en.cppreference.com/mwiki/index.php?title=c/memory/realloc&oldid=139021"