calloc

Defined in header `<stdlib.h>`

```
void* calloc( size_t num, size_t size );
```

Allocates memory for an array of num objects of size and initializes all bytes in the allocated storage to zero.

If allocation succeeds, returns a pointer to the lowest (first) byte in the allocated memory block that is suitably aligned for any object type.

If size is zero, the behavior is implementation defined (null pointer may be returned, or some non-null pointer may be returned that may not be used to access storage)

calloc 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 **calloc** 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 calloc. There is a single total order of all allocation and deallocation functions operating on each particular region of memory. (since C11)

Parameters

num	-	number of objects
size	-	size of each object

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().

On failure, returns a null pointer.

Notes

Due to the alignment requirements, the number of allocated bytes is not necessarily equal to num*size.

Initialization to all bits zero does not guarantee that a floating-point or a pointer would be initialized to 0.0 and the null pointer value, respectively (although that is true on all common platforms)

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

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

int main(void)
{
    int *p1 = calloc(4, sizeof(int));    // allocate and zero out an array of 4 int
    int *p2 = calloc(1, sizeof(int[4])); // same, naming the array type directly
    int *p3 = calloc(4, sizeof *p3);     // same, without repeating the type name

    if(p2) {
        for(int n=0; n<4; ++n) // print the array
            printf("p2[%d] == %d\n", n, p2[n]);
    }
    free(p1);
    free(p2);
    free(p3);
}
```

Output:

```
p2[0] == 0
p2[1] == 0
p2[2] == 0
p2[3] == 0
```

References

- C17 standard (ISO/IEC 9899:2018):
  - 7.22.3.2 The calloc function (p: 253)
- C11 standard (ISO/IEC 9899:2011):
  - 7.22.3.2 The calloc function (p: 348)
- C99 standard (ISO/IEC 9899:1999):
  - 7.20.3.1 The calloc function (p: 313)
- C89/C90 standard (ISO/IEC 9899:1990):
  - 4.10.3.1 The calloc function

See also

C++ documentation for **calloc**