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

int main()
{
    // Declare a pointer of integer type and allocate
    // (heap) memory for it.
    int *p = malloc(sizeof(int)); // type of the pointer
    // Initialise memory with some value
    // Because this code is compiled in C, the malloc
    // function implicitly will convert the void pointer
    // to an integer pointer.

    // Malloc only allocates memory.
    *p = 5;
    printf("%d", *p);
    free(p); // after this line, p still has a value, but it has a value of an invalid address. In this case after freeing the memory, p is pointing to an invalid address and this pointer is now known as a dangling pointer. A dangling pointer points to an invalid address and is very dangerous.
    // This is why it is a good idea to assign null to the pointer.
    p = NULL;


    // What happens if we try to free p again?
    free(p);
    // If you assigned p = NULL, this operation is ignored.
    // If this operation is an invalid address, your program will crash.
    your program will crash.
    // If we forget to call free, the memory that was allocated cannot be freed.
    // It cannot be released.
    // this results in a MEMORY LEAK. You lose the address to the memory that you've allocated. You can no longer release that memory. Memory leaks are a serious problem in c and C++ applications. This is why any memory allocated on the heap has to be freed manually by the programmer.

    /// Calloc
    // Accepts two arguments:
    // Argument 1: Number of elements to allocate
    // Size of element you woud like to store.

    // If we use calloc now,
    int *p = (int*) calloc(1, sizeof(int));
    // If we want to create and initialise 5 integeers
    int *p = (int*) calloc(5, sizeof(int));


    return 0;
}
```