How to Check Null in C

In C, NULL is a symbolic constant that always points to a nonexistent point in the memory. Although many programmers treat it as equal to 0, this is a simplification that can trip you up later on. It's best to check your pointers against...

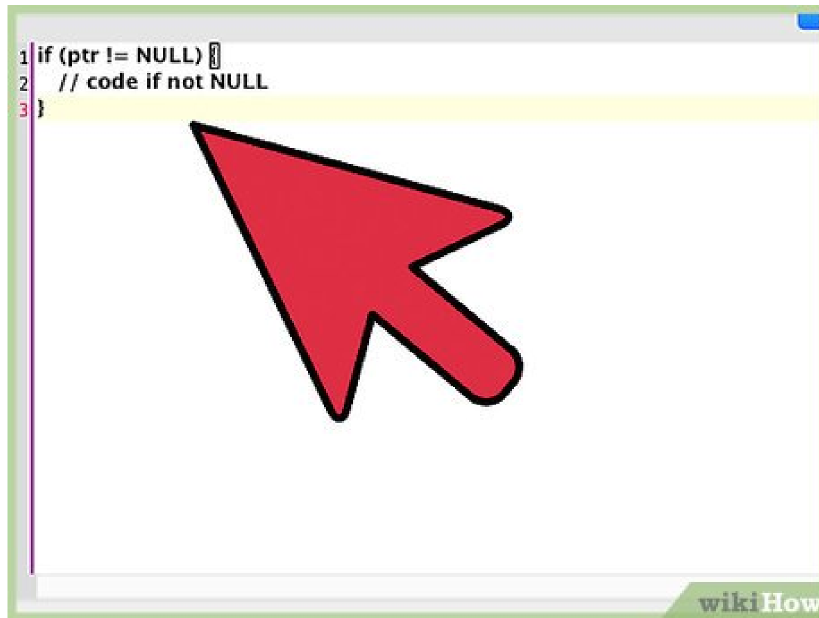
Part 1 of 2:

Performing a Null Check

1.

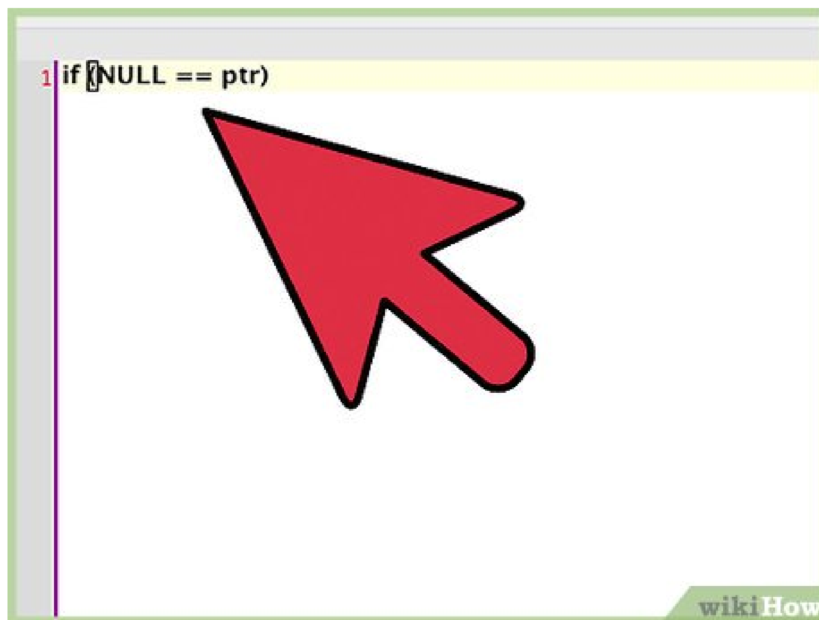
Use the standard null check code. The following is the most obvious way to write a null check. We'll use **ptr** in this article as the name of the pointer you're checking.

1. if(ptr == NULL)
```
{
  // code if pointer is NULL
} else {
  // code if not NULL
}
```

2. 

Test for any value but NULL. Sometimes it's more convenient to test for inequality instead. No surprises here:

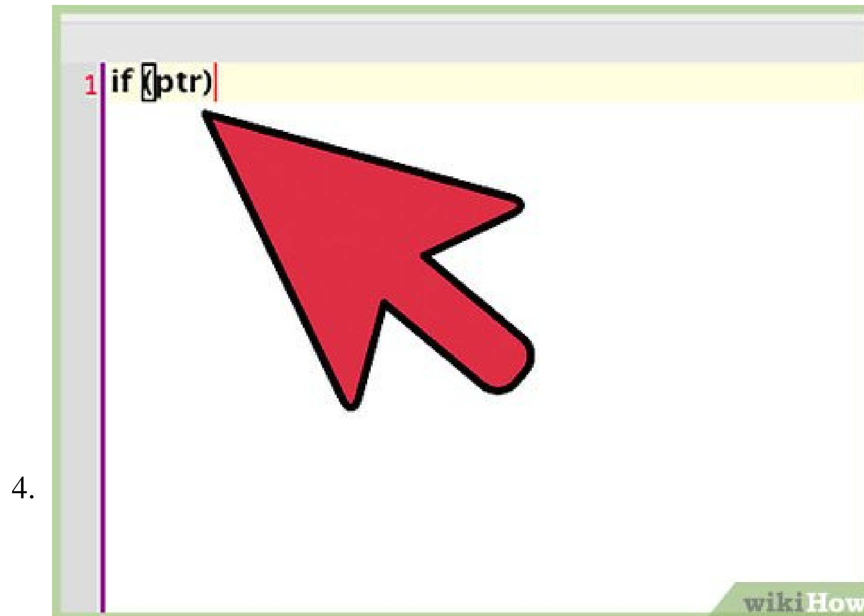
1. if (ptr != NULL) {
   // code if not NULL
   }

3. 

Write the NULL first to avoid errors (optional). The main disadvantage to the PTR == NULL method is the chance that you'll accidentally type ptr = NULL instead, *assigning* the NULL value to that pointer. This can cause a major headache. Since testing for (in)equality treats the operands symmetrically, you can get exactly the same result by writing **if (NULL == ptr)** instead. This is more typo-resistant, since an accidental NULL = ptr creates a simple compile error.

1. This looks a little awkward to some programmers, but it's perfectly valid. Which approach you use just depends on personal preference, and how good your compiler is at detecting the if (ptr = NULL)

error.

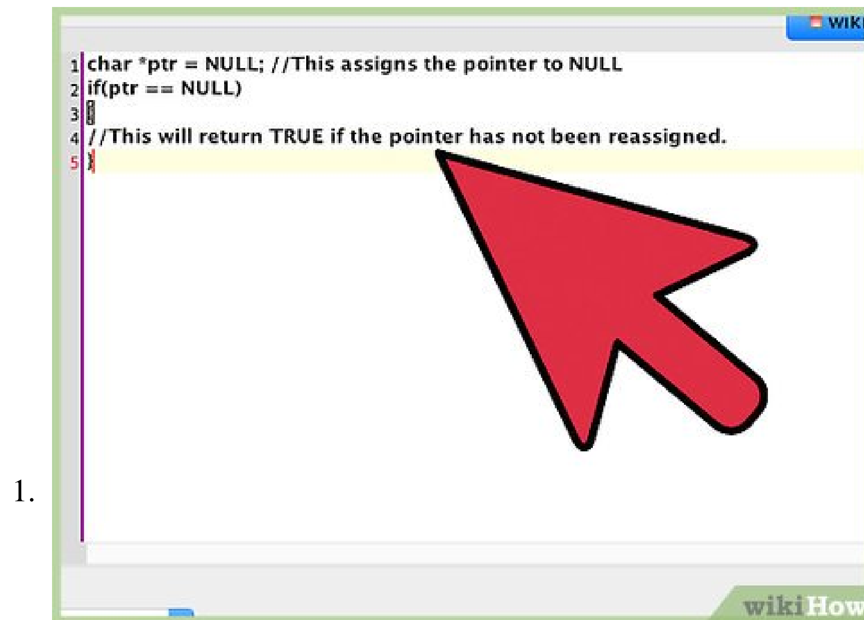
4. 

Test whether the variable is true. A simple **if (ptr)** tests whether ptr is TRUE. It will return FALSE if ptr is NULL, or if ptr is 0. The distinction doesn't matter in many cases, but be aware that these are not identical in all architectures.[1]

1. The reverse of this is **if (!ptr)**, which will return TRUE if ptr is FALSE.

Part 2 of 2:

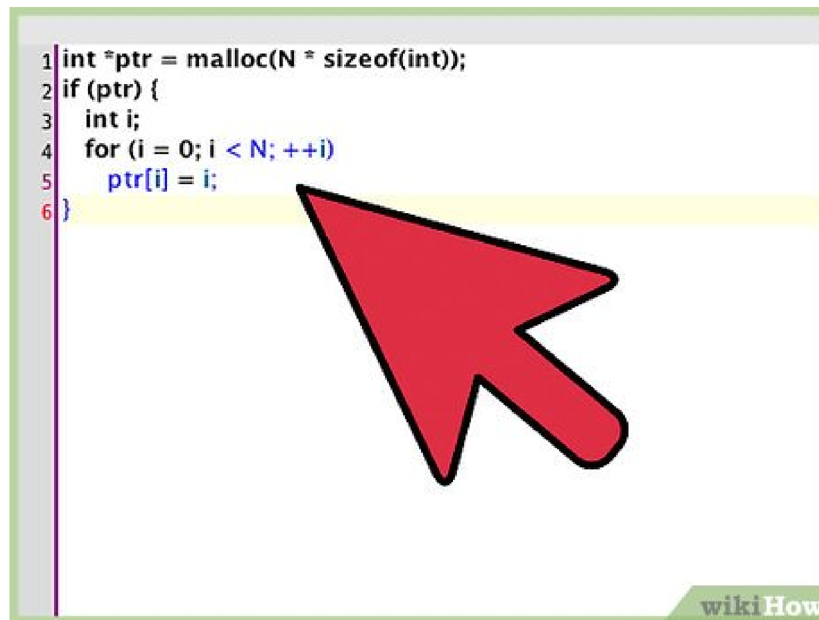
Avoiding Mistakes

1. 

Set a pointer before checking for NULL. One common mistake is to assume that a newly created pointer has a NULL value. This is not true. An unassigned pointer still points to a memory address, just not one

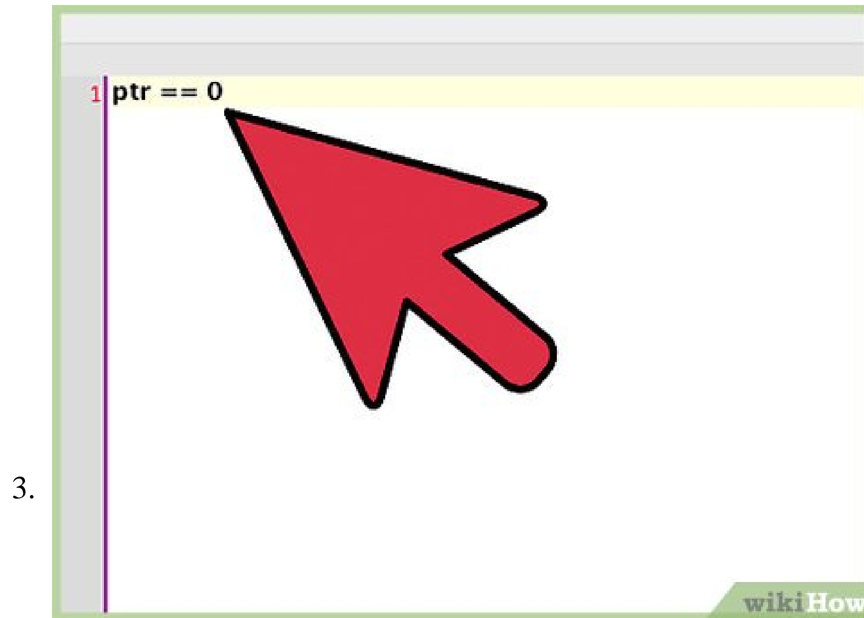
that you have specified. It's common practice to set newly created or newly freed pointers to NULL to make sure you don't use this unhelpful address by accident.

1. **Avoid this mistake:**
   char *ptr;
   if(ptr == NULL)
   {
   //This will return FALSE. The pointer has been assigned a valid value.
   }
2. **Instead write:**
   char *ptr = NULL; //This assigns the pointer to NULL
   if(ptr == NULL)
   {
   //This will return TRUE if the pointer has not been reassigned.
   }

2. 

Pay attention to functions that could return NULL. If a function can return NULL, think about whether this is a possibility, and whether that would cause problems later in your code. Here's an example of the malloc function using the null check (**if (ptr)**) to ensure it only handles pointers with valid values:

1. int *ptr = malloc(N * sizeof(int));
   if (ptr) {
     int i;
     for (i = 0; i N; ++i)
       ptr[i] = i;
   }

3. 

Understand that NULL is 0 but you should always use NULL instead of 0 when working with pointers for clarity. Historically, C represented NULL as the number 0 (that is, 0x00). Nowadays it can get a bit more complicated, and varies by operating system. You can usually check for NULL using **ptr == 0**, but there are corner cases where this can cause an issue. Perhaps more importantly, using NULL makes it obvious that you are working with pointers for other people reading your code.[2]

You finished reading the article "**How to Check Null in C**" edited by the [TipsMake](https://tipsmake.com) team. We hope this article has provided you with many useful tech tips and tricks. You can search for similar articles on tips and guides. Thank you for reading and for following us regularly.

© 2019 TipsMake.com