**Invalid pointers and null pointers.**

In principle, pointers are meant to point to valid addresses, such as the address of a variable or the address of an element in an array. But pointers can actually point to any address, including addresses that do not refer to any valid element. Typical examples of this are *uninitialized pointers* and pointers to non-existent elements of an array:

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 | int \* p; // uninitialized pointer (local variable)  int myarray[10];  int \* q = myarray+20; // element out of bounds |  |

Neither *p* nor *q* point to addresses known to contain a value, but none of the above statements causes an error. In C++, pointers are allowed to take any address value, no matter whether there actually is something at that address or not. What can cause an error is to dereference such a pointer (i.e., actually accessing the value they point to). Accessing such a pointer causes undefined behavior, ranging from an error during runtime to accessing some random value.

But, sometimes, a pointer really needs to explicitly point to nowhere, and not just an invalid address. For such cases, there exists a special value that any pointer type can take: the *null pointer value*. This value can be expressed in C++ in two ways: either with an integer value of *zero*, or with the *nullptr* keyword:

|  |  |  |
| --- | --- | --- |
| 1 2 | int \* p = 0;  int \* q = nullptr; |  |

Here, both *p* and *q* are *null pointers*, meaning that they explicitly point to nowhere, and they both actually compare equal: all *null pointers* compare equal to other *null pointers*. It is also quite usual to see the defined constant NULL be used in older code to refer to the *null pointer* value:

|  |  |  |
| --- | --- | --- |
|  | int \* r = NULL; |  |

NULL is defined in several headers of the standard library, and is defined as an alias of some *null pointer* constant value (such as 0 or nullptr).

Do not confuse *null pointers* with void pointers! A *null pointer* is a value that any pointer can take to represent that it is pointing to "nowhere", while a void pointer is a type of pointer that can point to somewhere without a specific type. One refers to the value stored in the pointer, and the other to the type of data it points to.

|  |  |  |  |
| --- | --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 | #include <iostream>  using namespace std;  int main()  {  int \*p = 0;  int \*q = nullptr;  int \*k = NULL;  cout << ((p == q) && (p == k) && (k == q)) << endl;  } | 1 | [Edit & Run](https://www32.cplusplus.com/doc/tutorial/pointers/) |