**The Differences Between a Pointer and a Reference**

Source: <http://stackoverflow.com/questions/57483/what-are-the-differences-between-a-pointer-variable-and-a-reference-variable-in>

1. A pointer can be re-assigned any number of times while a reference cannot be re-seated after binding.
2. Pointers can point nowhere (NULL), whereas reference always refer to an object.
3. You can't take the address of a reference like you can with pointers.
4. There's no "reference arithmetic’s" (but you can take the address of an object pointed by a reference and do pointer arithmetic’s on it as in &obj + 5).

To clarify a misconception:

*The C++ standard is very careful to avoid dictating how a compiler must implement references, but every C++ compiler implements references as pointers. That is, a declaration such as:*

int &ri = i;

***if it's not optimized away entirely***, *allocates the same amount of storage as a pointer, and places the address of i into that storage.*

***So, a pointer and a reference both occupy the same amount of memory.***

As a general rule,

* Use references in function parameters and return types to define useful and self-documenting interfaces.
* Use pointers to implement algorithms and data structures.

**Distinguish between pointers and references in C++**

Source: <http://www.cplusplus.com/articles/ENywvCM9/>

[Pointers and references](http://learnbyexamples.org/cc/distinguish-between-pointers-and-references-in-c.html) look different enough (pointers use the \* and -> operators, references use .), but they seem to do similar things. Both pointers and references let you refer to other objects indirectly. How, then, do you decide when to use one and not the other?  
First, recognize that there is no such thing as a null reference. A reference must always refer to some object. As a result, if you have a variable whose purpose is to refer to another object, but it is possible that there might not be an object to refer to, you should make the variable a pointer, because then you can set it to null. On the other hand, if the variable must always refer to an object, i.e., if your design does not allow for the possibility that the variable is null, you should probably make the variable a reference.  
"But wait," you wonder, "what about underhandedness like this?"

|  |  |  |
| --- | --- | --- |
| 1 2 3 | char \*pc = 0; // set pointer to null  char& rc = \*pc; // make reference refer to  // dereferenced null pointer |  |

Well, this is evil, pure and simple. The results are undefined (compilers can generate output to do anything they like), and people who write this kind of code should be shunned until they agree to cease and desist. If you have to worry about things like this in your software, you're probably best off avoiding references entirely. Either that or finding a better class of [programmers](http://learnbyexamples.org/) to work with. We'll henceforth ignore the possibility that a reference can be *null*.  
Because a reference must refer to an object, [C++](http://learnbyexamples.org/) requires that references be initialized:

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 | string& rs; // error! References must  // be initialized  string s("xyzzy");  string& rs = s; // okay, rs refers to s |  |

Pointers are subject to no such restriction:

|  |  |  |
| --- | --- | --- |
| 1 2 | string \*ps; // uninitialized pointer:  // valid but risky |  |

The fact that there is no such thing as a null reference implies that it can be more efficient to use references than to use pointers. That's because there's no need to test the validity of a reference before using it:

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 | void printDouble(const double& rd)  {  cout << rd; // no need to test rd; it  } // must refer to a double |  |

Pointers, on the other hand, should generally be tested against null:

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 5 6 | void printDouble(const double \*pd)  {  if (pd) { // check for null pointer  cout << \*pd;  }  } |  |

Another important difference between pointers and references is that pointers may be reassigned to refer to different objects. A reference, however, always refers to the object with which it is initialized:

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 | string s1("Nancy");  string s2("Clancy");  string& rs = s1; // rs refers to s1    string \*ps = &s1; // ps points to s1<a name="31186"></a>    rs = s2; // rs still refers to s1,  // but s1's value is now  // "Clancy"    ps = &s2; // ps now points to s2;  // s1 is unchanged |  |

In general, you should use a pointer whenever you need to take into account the possibility that there's nothing to refer to (in which case you can set the pointer to null) or whenever you need to be able to refer to different things at different times (in which case you can change where the pointer points). You should use a reference whenever you know there will always be an object to refer to and you also know that once you're referring to that object, you'll never want to refer to anything else.  
There is one other situation in which you should use a reference, and that's when you're implementing certain operators. The most common example is operator[]. This operator typically needs to return something that can be used as the target of an assignment:

|  |  |  |
| --- | --- | --- |
| 1 2 3 4 5 | vector<int> v(10); // create an int vector of size 10;  // vector is a template in the  // standard C++ library  v[5] = 10; // the target of this assignment is  // the return value of operator[] |  |

If operator[] returned a pointer, this last statement would have to be written this way:  
\*v[5] = 10;   
But this makes it look like v is a vector of pointers, which it's not. For this reason, you'll almost always want operator[] to return a reference.  
References, then, are the feature of choice when you know you have something to refer to, when you'll never want to refer to anything else, and when implementing operators whose syntactic requirements make the use of pointers undesirable. In all other cases, stick with pointers.