Certainly it’s true that

1. p->a

is equivalent to

1. (\*p).a

But I’d add some more detail. First of all, p is always a pointer (or pointer-like entity, such as an iterator) in this context. In C, a pointer might very often point to an instance of a structure; in C++, a pointer might often point to an object. It is so common to use pointers to structures and objects, the “arrow” operator (->) was added to the language to save a few keystrokes for this pointer-dereference-and-access operation.

And yet this saves only a few keystrokes, perhaps not enough to justify its inclusion in the list of C/C++ operators. The real reason for its inclusion its that in certain programs, it’s very common to have linked-list and tree structures, such as this:

1. struct Node {
2. int val;
3. Node \*next;
4. } \*root;

Given such a definition, it’s common to reference the next node in the list, and the next after that, etc., so that you might have code like this:

1. root->next->next->val = 100;

This means: access the third item in the linked list, and assign the value 100. This code is definitely more elegant and readable than the following, quite apart from number of keystrokes that are saved.

1. (\*(\*(\*root).next).next).val = 100;