STRO84
Stroustrup, Bjarne
Operator Overloading in C++
AT&T Bell Laboratories
Murray Hill, New Jersey 07974
1984

In this paper, Stroustrup explains the notion of operator overloading, especially as it applies to his extensions to the C language, C++. In C++, classes provide the programmer with the capabilities necessary to define non-primitive objects, together with the operations to be performed on these objects. Using conventional operators and defining semantics for each one, is a more convenient and "natural" way of representing the manipulation of these objects. The paper does assume some familiarity with C++; it is an in-depth study of one of the more important features of the language. He begins by motivating the subject in general by showing how one would define the common arithmetic operators on a newly-defined class of complex numbers. He then explains the basic aims of the design of the operator overloading mechanism. There are 4 of these: Data type definition must be as "elegant" as with primitive data types. These operators must be implemented efficiently. The base language must be immutable - not subject to change by the programmer. Compilation of programs using the overloading facility must be relatively easy to compile. Unfortunately, he does no more than state his goals; he does not explain whether or not he has met these goals. Presumably, it is left for the reader to judge. He then uses the example of a class called complex numbers, so chosen because a complex number has a trivial representation, a set of known operators, and a conventional notation for these operations. He shows how to specify rules for initialization and for type conversion. In explaining how his operator overloading provides this conventional notation for the programmer, he must explain some of the constructs of C++. He does this very briefly - this is why prior familiarity with the language would be helpful. Next he shows how operator overloading would apply to less trivial data types, such as matrix or string. He explains this well and here it is clear that operator overloading clearly has its advantages. In elaborating on these, he introduces two other C++ features, references ("pointers" to objects of a class) and, for strings, free storage management and the sharing of objects. He attempts to show the runtime efficiency of using the overloading facilities and explains some techniques used in implementing them in C++. The article is well written and in general, accomplishes what it sets out to do. It is too bad that he doesn’t give some support for the notion that his design has met his stated goals. As a study of a single language feature it is reasonably thorough and complete.