STRO84
Stroustrup, Bjarne
Data Abstraction in C
Computing Science Technical Report No. 109
AT&T Bell Laboratories, 1984

In this paper, Stroustrup introduces C++, a superset of the C language. It is an extension to the original language, offering various data abstraction facilities. The paper is intended as a guide for the user wanting to write C++ programs. Each extension to C is discussed and (short) examples given. It is also attempts to provide the programmer with an indication of the programming techniques C++ supports and what errors it is intended to help avoid. At one time known as "C with classes", C++ evolved through a few different incantations. Its primary stated influence was the language Simula. The idea was to incorporate data abstraction principles into a language known for its efficiency and terseness so as to aid in the production of large software systems. Great care appears to have been made to ensure that the extra features add little if any overhead in terms of space and time compared to "old C". (In fact, the C++ compiler is claimed to be more efficient thatn the C compiler). Also, an effort was made to truly implement a superset of C in the sense that old C programs could be compiled and run using the C++ compiler. (This was largely successful - there are only a few minor inconveniences associated with moving a C program to C++.) In short, it is hoped that C++ will provide the best of both worlds to a programmer - higher level data-hiding capabilities to make it easier to model a given domain and low-level C constructs when it is necessary to express low-level detail. After initially defining data abstraction and justifying its desirability, Stroustrup covers the major features of his extensions to C. These include:

restriction of access to data -
- class (similar to Simula).
- the concept of an object (loosely, an instance of a class).

constructors and overloaded functions -
- more "elegant" initializers and "creators" of objects (i.e. allowing default arguments and types).
- dynamic typing (ie. selection of a function depending on the type and classs of its arguments.

operator overloading -
- user defined operators (only allowed for pre-defined C operators).

derived classes -
- a form of specialization of a function. (As mentioned above, the selection of the function to call depends upon the type and class of its arguments).

polymorphic functions and classes
- "generic" functions and classes.
- functions that are so designed so that they can operate on different argument types.
- classes can be designed without concern for kind of data structures programs using them will need.

input and output -
- a partial solution to this problem area is suggested.

At the end of the paper, Stroustrup includes a discussion of efficiency, implementation and compatibility issues. Finally, he concludes with a quick comparison with other languages.

The paper is well written, although a little short on illustrative examples. (If this is meant as any kind of tutorial, then it presumes too much familiarity with the subject in my opinion). As far as C++ goes, my initial reaction is: "Why bother?". It seems a little like trying to dress the carpenter in King's clothing. C seems aptly suited to the task for which it was designed and any extension would have to extend that suitability rather than trying to make it into something it never was intended to be. While Stroustrup claims that his extensions are designed to help the programmer with large scale software systems, it is not entirely obvious that this has been successful. (Perhaps a further paper showing a real application of C++'s features would be in order now.) As far as how well it does what it says it does - I have the following comments about the language:

- There is a little real little semantic constraint placed on the programmer. In some sense, C++ seems like a large scale macro package, with some compile time type checking thrown in. This would be okay, if the semantic checks were constraining and treated consistently. For example, function overloading (one of C++ major strengths, I feel) is purely syntactic - the compiler attributes no semantics to this. Also class inheritance is very easily subverted or ignored, thereby weakening the abstraction mechanism.

- Because of this lack of semantic constraints, the responsibility is placed on the programmer to maintain the consistency of his data model. He must be very disciplined in the sense that the language only provides the tools and does little to enforce consistency of their use.

- Also, separate compilation without an intelligent loader (one that knows about types and classes), makes it too easy to defeat the semantic constraints.

- Classes are 2nd class objects - are they even objects?

- I/O is awkward at best in C++ due to the lack of pre-defined functions for this.

- C++'s strength, in my opinion, lies mainly in two areas: function typing and operator overloading. Classes are a nice follow-up of these, but it seems to me that C's "model of computation" does not sit too well with this concept.

C++ is an interesting attempt to add to the usefulness of C. I'm not sure at all that it has succeeded even though it has contributed a few desirable features. I suppose the test of usefulness may be in the market-place. I do wonder how widely (if at all) it will be accepted.