WG14/N368

X3J11/94-053

Why both C and C++ standards are necessary

C++ started life as a language designers experiment with something later to be called object oriented programing. The use of an existing language, C, as both a basis from which to create a new language and as the high level assembler into which C++ was translated was seen as having many advantages. This, original, close association has resulted in the two languages becoming intertwined in the publics mind.

The original driving force behind C was the feeling that languages ought to be small, easy to implement and produce efficient code. It was, to some extent, more influenced by the hardware of its day than some high level software design methodology.

The concept of object oriented programing is still only a few years old. Because of this there has been very little practical experience in designing and building applications based on this methodology. Knowledge on how to design languages, to support such designs and implementations, lags even further behind.

Through various influences C++ has become a large complex language. Its priority being to make life simpler for the software developer in the creation of reusable libraries and competitive applications (that is large applications containing a lot of functionality).

C is a small language. Its users often view efficiency and the ability to get close to the hardware as being important. As such the language does not contain any constructs that might unexpectedly impose a high runtime overhead or be difficult to map onto the commonly available hardware.

Many software developers have moved from C to C++. There are many reasons for this shift. I shall leave it to future historians to document why this moved occurred.

With the C standard soon due for reconsideration it is necessary to address the belief that the two languages are seen as being strongly connected to each other.

The C panel has discussed its relationship with C++ several times. The it has always unanimously agreed that the two languages are connected, but separate. Each deserving their own standard.

The reasons for this are as follows:

1) C is a small language, capable of generating efficient code. This makes it very suitable for use in embedded system, where generated code size and quality are important issues.

5

2) In number crunching applications speed is very important. C currently looses out to Fortran in the degree to which it can be optimised. This issue is being addressed by the ANSI Numerical C Extensions Group. It is expected that with suitable minor additions to the languages it will be able to compete with Fortran in compute intensive areas.

3) C++ is a large language with a lot of hidden runtime overheads. It can be very difficult to predict what code might be generated for use of relatively simple constructs in the source.

4) The C++ standardisation effort is learning that some of the early language design decisions, taken with C compatibility in mind, are having an adverse effect on the creation of a coherent C++ language. In order to best serve the community of C++ users WG21 may decide to make further fundamental changes. Any attempt by WG14 to mirror such changes, to accommodate a view of the world that C does not have is likely to lead to significant compatibility problems with existing code.

5) The view that C is currently a true subset of C++ is not correct. C++ has evolved and continues to evolve away from C. That this view has common currency has more to say about the state of the average programmers knowledge of the two languages than about what is contained in the two defining documents.

At its last panel meeting the committee discussed what it saw as the future of C. It was felt that there should be some additions to the language to aid its role as a high level assembler. Perhaps also some minor changes to accommodate C++, where there was little likelyhood of breaking existing code. It general it was agreed that C should stay as a small language and that nothing was to be gained by merging it into C++.