WG21/N2014 J16/06-0084 2006-04-21

Nick Maclaren
University of Cambridge Computing Service,
New Museums Site, Pembroke Street,
Cambridge CB2 3QH, England.
Email: nmm1@cam.ac.uk
Tel.: +44 1223 334761
Fax: +44 1223 334679

C99 and POSIX(2001) Compatibility

Introduction

This proposal is the consequences of attempting to maintain compatibility with these. It assumes that there has been a decision to do so, but not at the expense of harming C++. If that is wrong, then it becomes obsolete.

To a first approximation, everything in this document is specific to the library. The `long long` issue is referred to elsewhere, as will be the "IEEE 754" ones (except as an example).

It proposes one change in principle and one in detail, that I believe would enable adequate compatibility with C++(1998), C99 and POSIX(2001), even when threading is added to C++.

Proposal

1. Facilities included via the C++ headers (e.g. <`cstdlib`>) should maintain compatibility with C++(1998), and those included via the 'native' C headers (e.g. <`stdlib.h`>) should track C99 and POSIX(2001).

To enable this, some constraints will need imposing on when an object can be created in C++ and used in C (and vice versa), especially when accessed from separate threads without explicit synchronisation. All such constraints will be proposed in the normal way.

2. To maintain compatibility with C++(1998), the C++ specification of <`math.h`> should say that `math_errhandling` must include `MATH_ERRNO` — i.e. it must have value either `MATH_ERRNO` or `MATH_ERRNO|MATH_ERREXCEPT`.

Justification

1. Both C99 and POSIX(2001) have changed specifications in ways that fit very badly with C++, and where the consensus seems to be that C++ does not want to follow them.

The most obvious is POSIX's specifications of its threading facilities, including the memory model, thread-safety and termination. It is extremely unlikely that it will be possible to make C++'s threading facilities fully compatible with POSIX, without compromising C++'s design objectives. A simple example is that POSIX states that all C99 I/O functions must be thread-safe (even when used on a single file), which is incompatible with an efficient, scalable implementation.

There are similar problems in C99, especially in the "IEEE 754" support; I intend to describe some of these elsewhere, but one example is worth mentioning. C99 (7.6, paragraph 2 and 7.6.1, paragraph 2) says that, for any code not compiled with the `FENV_ACCESS` pragma on (effectively including all library functions not in <`fenv.h`> and <`math.h`>), all floating-point control modes must be in the default state on entry, and all floating-point status flags are unspecified on exit. That includes `malloc`.

That makes some kind of sense in C99, where the library is called only implicitly, but it does not in C++ (see footnote 32 in C++(1998)!) The effect of the C99 wording is that effectively all use of <`fenv.h`> facilities leads to undefined behaviour in C++.

So it is very likely that the C++ library may need slightly different semantics from the C99 one. I believe that the best way to proceed is to extend the current separation of facilities included via the C++ headers (e.g. <`cstdlib`>) from the 'native' C ones (e.g. <`stdlib.h`>), and to maintain C++ semantics for the former, and let the latter drift with C99.

This may sound a trivial point, but has significant specification and implementation consequences. In particular, it is essential to forbid C++ and C library facilities being used on the same object in two threads, without suitable synchronisation. I believe that the only two areas that are seriously affected are locales and I/O, but I believe that locales could be defined to be thread-safe without an unreasonable performance penalty.

2. C99 has made several changes that are incompatible with C89 and C++; most of them will affect only a few users and implementations and I ignore them.

A more serious problem is that it permits incompatible, implementation-selected language variants of C99, some of which are incompatible with C89 and C++. Note that I am not referring to simple optional extensions, but to cases where it is hard or impossible to write code that will work under all C99 compilers, and where existing, important, portable, conforming C89/C++ programs are now undefined.

The most important is that `long` was guaranteed to be the longest built-in signed integer type in both K&R C and C89, but is not in C99. I describe that issue elsewhere.

Another one has recently been the subject of a debate on the `c++std-lib` mailing list. C99 permits an implementation not to set `errno` in <`math.h`>, which means that any existing C89/C++ code that relies on `errno` being set in <`math.h`> will now fail, quietly on some implementations. Equally seriously, it forces programmers to write dual-mode error detection if they want both full portability and maximum robustness.

My belief is that incompatible, implementation-selected language variants should be avoided if at all possible, and that compatibility with C++(1998) should be preserved. Where extensions are essential, they should be such as to be compatible with the unextended language.