N20?? Input & Output of NaN and infinity for the C++0x Standard Library

Document number: JTC 1/SC22/WG21/N20??
Date: 2006-05-15, version 1
Project: Languages C++
References:
1 C++ ISO/IEC IS 14882:1998(E),
2 William Kahan http://http.cs.berkley.edu/~wkahan/ieee754status/ieee754.ps

Reply to: Paul A Bristow, pbristow@hetp.u-net.com, J16/04-0108, www.hetp.u-net.com

Contents

1 Background & motivation

Why is this important? What kinds of problems does it address, and what kinds of programmers is it intended to support? Is it based on existing practice?

Although C++ limits library provides values for quiet_NaN and infinity, and C99/ C++TR1 provides a mechanism for testing if values are finite or infinite or NaN, there is no **Standard** mechanism for streaming input and output of these as decimal digits strings. This has impeded the potential use of NaN as a way of storing 'unknown' or 'missing' values. These values may exist either because of some computation problem in their evaluation, or because they were never measured or their input left blank. Similarly, the usefulness of Infinity has been much reduced because it is not possible to output or input them.

This lack has been highlighted by use of the Boost.Serialization and Boost.Lexical_cast libraries when the facility of storing both NaN and infinite values has been much missed (compared to the invaluable use of NotADate by the Boost.DateTime library).

Some iostream implementations provide a string representation output as a result of, for example:

```
std::cout << std::numeric_limits<double>::quiet_NaN() << std::endl;
```

but

```
double v;
std::cin >> v;
```

fails to assign the hoped-for NaN to v (and worse, may assign a misleading valid numeric value to v!).

This proposal is prompted by the addition of C99 functions isnan and isfinite to C++: these now provide a portable and standard method of checking whether floating-point values are NaN or infinite, something that was (regrettably) not possible before.

2 Impacts on the C++ Standards

This is a pure addition to existing the iostream class and numeric_limits class. It does not require any language change, nor any change to the existing values provided by numeric_limits. It may be sensible to add additional indications that an implementation provides both input and output of these standard strings (it is difficult to see why either one of these should be provided without the other). It will not change the behavior of existing programs that only stream finite floating-point values.

3 Design Decisions

How to detect NaN and infinities?

It is proposed to rely entirely on the C99 functions isnan and isinf to detect Not-A-Numbers and Infinities respectively. This places the burden of deciphering the floating-point format on the C99 library.

Signaling NaNs?

Signaling NaNs, if detected by isnan, are assumed to be treated exactly like quiet NaNs, and after 'round-tripping' by output to a string or file and re-input will re-appear as numeric_limits<>FPT>::quiet_NaN.

How to detect the ability to handle streaming of NaNs and infinities?

I propose adding to numeric_limits, to indicate the presence of (new) code to handle streaming of NaNs and infinities, the following

```
static const bool has_NaN_io = true;
static const bool has_infinity_io = true;
```

Obviously, if the type is not specialized for the type, or if the streaming input and output of NaNs and infinities are not implemented (yet), these values will be false: this allows existing programs to execute exactly as at present.

How to allow programs that can handle NaN and infinity IO to adopt their previous behavior?

An optional implementation defined macro would seem the obvious way of permitting programs which rely on their previous behavior to work as with current IO libraries.

Hexadecimal format?

Although it has been tempting to consider a hexadecimal representation, so that all possible floating-point representations (including all NaNs and infinite and finite values) can be output and input, this is inherently dependent on the floating-point size, and floating-point layout, and thus much less portable.

So for simplicity, I therefore propose a single string representation for all floating-point types and all sizes and all layouts, including byte order or endianness.

Existing systems

Microsoft provides an output for infinity as "1.#INF",
but this string is NOT read as infinity on input, but as "1" and, unhelpfully, gives no indication from the stream state that this value may be misleading. Similarly for NaNs, there are no clues on re-input, unless a program checks the terminating characters for the NaN warning.

VisualAge C++ in contrast does handle both output *and input* of NaNs and infinities as strings "NAN" and "INF" whose case can be controlled for "nan" and "inf", but is ignored on input.

The Dinkumware library also uses these C99-compliant strings.

GCC ???

Standard NaN and infinity strings, or variable?
Some systems might want to print "missing" or "undefined", something that is already possible by changes to num_put, but it is my belief that Standard strings, at least for the classic C locale, would be much more widely useful because, at least for Boost.serlialization and Boost.lexical_cast, portability is most important, especially as very many systems have and use 64-bit doubles whose decimal 17 digit string representation is the same, even if the internal layout is not.

Same string for all types, sizes and layouts?

It is also necessary to consider the string for all NaNs and infinities should be the same for all floating-point types, float, double and long, and also for User Defined Floating-point types, for example, the NTL classes, quad_float (128-bit with a 113-bit significand) and class RR, offering an arbitrary precision, typically 100 decimal digits.

For simplicity, it is proposed to only consider a single (signed) NaN that is defined by numeric_limits<FPType>::quiet_NaN, and a single (signed) infinity value.

```
if (finite) // Normal floating-point value – neither NaN nor infinite
{ // C99/C++ TR1 detection of (non-)finiteness.
   // Output normal decimal digit string (usual formatting).
}
else
{ // Not finite.
   if (isnan())  // C99/C++ TR1 detection of any of the many possible NaNs.
   { // is NaN.
      // Output some Standard string,
      numpunct.NaN_name(); // ? see below.
```

```
            // for which "NaN" would seem simple, concise and short.
        }
        else
        { // Is infinity.
          // Output some Standard string,
          numpunct.infinity_name();   // ? see below.
          // for which "Inf" would seem simple, concise and short.
        }
    }
```

It is difficult to see any reason to make the strings for NaN and infinity locale dependent, for example in different languages, if their use is likely to be internal.

But some systems might want to print a locale or specific string such as "missing" or "undefined". This is not too difficult, but they will have to deal with the more difficult problem of recognizing these strings on *input*.

Where to store the string representation of NaN and infinity?

To provide this, I suggest adding
```
  string_type NaN_name const;
  string_type infinity_name const
```

 to std class numpunct after decimal_point... truename and falsename.

What strings should represent NaN and infinity?

C99 already specifies a range of output formats for nans and infs with %f:

ISO/IEC 9899:1999 (E) 7.19.6.1p8:

"A double argument representing an infinity is converted in one of the styles
[-]inf or [-]infinity - which style is implementation-defined. A
double argument representing a NaN is converted in one of the styles
[-]nan or [-]nan(n-char-sequence) - which style, and the meaning of
any n-char-sequence, is implementation-defined. The F conversion specifier
produces INF, INFINITY, or NAN instead of inf, infinity, or nan,
respectively"

And input formats in 7.20.1.3 (The strtod, strtof, and strtold functions):

"- one of INF or INFINITY, ignoring case
- one of NAN or NAN(n-char-sequenceopt), ignoring case in the NAN part,
where:
n-char-sequence:
  digit
  nondigit
  n-char-sequence digit
  n-char-sequence nondigit"

It seems only sensible to adopt this, in that C++0X will presumably incorporate this.

Should the strings be upper, lower or mixed case?
Some systems appear to allow both upper and lower case output, but not the more attractive mixed case "NaN" or "Infinity".

It is hard to see any reason why the case should not be ignored on input.

What strings for NaN and inf?
As concise, I propose "NaN" and "inf" because they are:
simple.
short.
same length (less layout problems).
reasonably language neutral

Sign?
It is proposed to handle sign entirely separately in the existing standard way.
This allows for positive and negative infinity, and for both positive and negative quiet_NaN.

Is NaN the best way of indicating 'missing' values?

http://babbage.cs.qc.edu/courses/cs341/IEEE-754references.html

IEEE754 has defined an "Indeterminate" sign 1 exponent 11..11 10..00 and significand FFC00000.

Or as a double big-endian 0xfff8000000000000.

Would this be a better choice to indicate that a value is 'missing'?
(Bearing in mind that NaN is usually the result of a computation rather than just 'never existed'.

Draft of Proposed Revised Text for

18.2.1.1 template class numeric_limits

TODO!