Doc: X3J16/96-0101=WG21/N0919
Date: May 26, 1996
By: Philippe Le Mouël

Instantiation of iostreams classes on base types

Description

The current WP does not allow instantiation of the iostreams classes on base types. The problem is located in *basic_ostream* and *basic_istream* classes. *basic_ostream* overloads insertors on both the base types with the exception of *char* and *wchar_*t types, and the character type on which the stream is instantiated. Therefore, whenever a stream is instantiated on a base types other than *char* or *wchar_t*, a conflict arises between the insertor of this type and the insertor of the character type. The same conflict arises for *basic_istream* and extractors.

Note: This paper follows the one on “Insertion and Extraction of *char*, *signed char* and *unsigned char*”.

Discussion

If we want to instantiate a stream on one of the base types, we have to decide whether we want to treat the base type as a character or as a numeric value. Jerry Schwarz pointed out two reasons to prefer interpretation as a numeric value. “The first one is the frequency of occurrence, and the other is how easy it is to get the other interpretation.” If we choose interpretation as a numeric value, we can get character behavior by using the unformatted functions, but if we chose interpretation as a character, we do not have the other option. Instantiating iostreams classes on base types, and treating insertion and extraction of values of this type as numeric values, can be implemented by removing all the insertors and extractors from *basic_ostream* and *basic_istream* respectively and adding the following global template functions:

```
template <class charT, class traits>
basic_ostream<charT, traits>&
operator <<( basic_ostream<charT, traits>& out, charT c)
{ ; }
```

Treat as a character (see Insertion and Extraction of *char, unsigned char and signed char* paper).

```
template <class charT, class traits>
basic_ostream<charT, traits>&
operator <<( basic_ostream<charT, traits>& out, int c)
{ ; }
```

Treat as an int.

```
template <class charT, class traits>
basic_ostream<charT, traits>&
operator <<( basic_ostream<charT, traits>& out, short c)
{ ; }
```

Treat as a short.

This pattern continues for all the base types, and the same applies to *basic_istream* and extractors as well. Therefore if you do the following:

```
basic_ostringstream<short, char_traits<short> > out;

short b = 56;

out << b;
```

you will call:

```
template <class charT, class traits>
basic_ostream<charT, traits>&
operator <<( basic_ostream<charT, traits>& out, short c)
{ ; } // treat as a short
```

Which is more specialized than:

```
template <class charT, class traits>
basic_ostream<charT, traits>&
operator <<( basic_ostream<charT, traits>& out, charT c)
{ ; } // treat as a character
```

Allowing instantiation of iostreams classes on base types is only part of the story. The iostreams library relies on locale components such as *ctype* and *codecvt* facets to perform its job. Furthermore, the default locale imbued in both the stream and the stream buffer provide facets for only *char* and *wchar_t* types. So in any case, we will not solve users problem when instantiating iostreams classes on base types by only accepting the changes described above. Users will have to provide the following components for every base types they want to instantiate iostreams on:

- class *char_traits* specialized on the base type
- locale *ctype* facet specialized on the base type
- locale *codecvt* facet specialized on the base type
- locale *numpunct* facet specialized on the base type

Another approach, is to wrap the base type in a class and to use it to instantiate the iostreams classes. There are two major advantages with this scheme: first, we do not need to change a single line of the WP, and users can get both numeric and character behavior when using insertors and extractors. The price to pay for users is only a few more lines of codes which can be neglected in comparison of the number of lines they will have to write to provide the components required by iostreams.

Proposed resolutions

1. Do not allow direct instantiation of iostreams classes on base types. In this case users will still be able to wrap the base type in a class and to instantiate iostreams with this wrapper class. With this solution, users will have to do some extra work, but nothing compared to providing the right *char_traits* and locale components required by iostreams. This solution also makes it possible to keep both the numeric and the character behaviors of insertor and extractor.

2. Allow instantiation of iostreams classes on base types as described above.