Unification of Traits

Norihiro Kumagai
Sharp Corporation

96.1.30

1. Introduction

There are several problems in the specification of character traits in the "26 Sep. 1995" WP(95-0185/N785).

(1) The specification is complecated because of there are two different character traits, 'string_char_traits' and 'ios_traits'. We have to pass two different traits to some member functions of string class handling iostream.

Both 'string_char_traits' and 'ios_traits' has almost the same role to handle a set of character container type and its related functions. There are a lot of member functions whose functionality is overlapped between both of the traits.

So we propose to unify both of the traits, to define single character traits so as to provide the second template parameter for string/iostream classes. This leads us simpler member function interfase of the string class which handle iostream.

(2) We cannot specify two or more different int_type, pos_type, off_type, state_type definitions to the ios_traits. Each of them is fixed to an implementation-defined type for any different character container types. Because we really need to some method to specify two different int_type in two cases the character container type is char and wchar_t. There need some inprovement of the specification of the current WP.

Some Japanese members have pointed them out (95-0210/N0810, 95-0064/N664). This document propose a specification of unified traits, reflecting some of the results of discusson in Tokyo (Nov. 95).

2. Approach

[Basic Discipline]

* No more new functions/features are added. Otherwise, some functions have disappeared in the result of simplify.

* I have give more clearer description, especially for constraints upon the character container type, 'charT', and relation between traits and character container type during specialization.

* All of the locale-dependent functions are removed. Implementation is expected to invoke some character classification fuctions (for example, is_whitespace in ios_traits) in the locale object imbued to the iostream object.

[Approach]

We can classify all of the types and member functions in string_char_traits and ios_traits into the following categories.

I) character container type, charT and the functions whose signature
   has no other types/classes than 'char_type'.

```
     type:      char_type
     functions: assign, eq, ne, lt, compare, length, find, copy, assign.
```

II) functions whose signature contains other types/classes than
    'char_type'.

```
     not_eof, to_char_type, eq_int_type, get_state, get_pos.
```

III) defined types, int_type, state_types, pos_types, and off_types.

IV) character constant functions, eos, eof, newline.

V) locale-dependent functions, is_whitespace.

Almost all functions in category (I), most of which are from string_char_traits, are needed in iostreams, so thery are expected to consist of minimul set of the trais member functions. A template struct 'char_traits' is for holding such the functions.

Different definitions of functions either in category (II) or in (III) shall be provided for each of the different character container types. They are not in the template definitions of 'char_traits' but shall appear in the each specialization of the traits.

For each of the types and member functions, I have made it clear whether it shall be referred in string class or not, and whether in iostream class or not. This cause to make it easy to provide new character container type. For example, in case we want to provide a character container type and its related traits, all we need is to define the types and functions which referred only in stream class only. There is no need to take any care about types and functions for iostreams.

In category (IV), both of the eos, newline functions have their corresponding single-byte characters, '\0', '\n', respectively; so it can be represented as char_type(0), char_type('\n'), if every character container type shall have integral constructor. Note that the meaing of both of the characters are locale-independent so we cannot rely on the locale class.

The function, 'is_whitespace', originally introduced to provide flexibility of the definition of whitespace by preparing it in traits. Discussion in Tokyo meeting reveals that it cause some inefficiency to skip whitespaces. The locale class has now provided a set of member functions for skipping whitespaces, so the whitespace testing function in the traits is eliminated with no fear for lackness of such the functionality.

3. Changes made in this specification

The following is the exception for the rule not to change any function signature;

1) [call-by-value interface] There is some inconsistency about the
   character argument in function signatures. Those in
   string_char_traits adopts pass-by-reference, on the contrary, those
   in ios_traits specifies call-by-value.

I have decided all of the functions pass by value. Because usual character container types are expected to be small and value-oriented, copying in invokation of the functions cause not so much performance degradation.

2) [integer constructor for character container types] In order to eliminate functions returns some character constants, such as eos(), eof(), newline(), we need functionality to convert some character (or integral) constant, such as '\n', '\0', EOF, into such constant value of the container type.

3) [eliminated function list]

   is_del is removed according to the discussion in Tokyo meeting.

4. New specification

The description of types and functions in traits common to both string and iostream is appended in clause 20(General utilities library), so template definition of char_traits and two of its specialised versions are defined in the header <utilities>.

The specific features to each of string/iostreams are described in top of each clause, that is, clause 21(strings) and clause 27(iostreams).

In following I will show an image of new specification about traits.

In this description, the box surrouned with '#' characters express change request in some specific area in the WP. I hope these box will not appear in the next version of WP.

```
------------------------------------------------------------
######################################################################
## the second paragraph in 17.2.1.2 Requirements [lib.structure.requirments]
## rewrite the former description to latter
######################################################################
```

From:

```
<  The string and iostream components use an explicit representation
< of operations required of template arguments.  They use a template
< class name XXX_traits to define these constraints.
```

To:

```
>   The string and iostream components use an explicit representation
> of operations required of template arguments.  The operations
> required are defined as the constraints of the template class,
> std::char_traits, specified in Clause 20(20.5), 21(21.1.1.1), and
> 27(27.4.2).
```

```
------------------------------------------------------------
######################################################################
## rewrite second paragraph in [lib.utilities]
######################################################################
```

From:

```
< The following subclauses describe utility and allocator
```

< requirements, utility components, function objects, dynamic memery
< management utilities, and date/time utilities, as summarized in
< Table 37.

To:

> The following subclauses describe utility and allocator
> requirements, utility components, function objects, dynamic memery
> management utilities, generic requirements of character traits, and
> date/time utilities, as summarized in Table 37.

##
[lib.utilities.traits] Subclase 20.5 should be appended as follows;
##

20.5 Character traits

This subclase contains a template class for representing {\it
character traits}, 'char_traits' and two specializations ('char' and
'wchar_t' versions) of it.

Most classes specified in each of the clause 21(string), and clause
27(iostream) need a set of defined types and functions to ensure their
behavior. These types and functions are provided as a set of defined
types and static member functions in the template parameter, 'traits'.
This subclause also describes these functions with the information
which library needs it.

In order to specialize for generating an template string or a template
iostream class to handle a character container type, CHAR_T, we can
specify it and its related character traits, TRAITS_T, as a pair of
the template parameters, charT and traits.

The struct TRAITS_T represents a character traits related to the
character container type, holding one or more types and static member
functions to be referred in the implementation of such the template
classes. In order to ensure behavior of objects of the specialized
classes, CHAR_T shall satisfy certain requirements and TRAITS_T shall
holds the set of types and member functions which satisfy other
requirements, these requirements depends on the cagegoly of template
classes (for example, strings and iostreams).

In this subclase, the requirements common to all of the template
classes whose template parameters are 'charT' and 'traits' are
described. Those which specific to a certain category of template
classes are described in each clause (in clause 21 for strings, clause
27 for iostreams).

In this subclase, also, a template struct of character traits
'char_traits' which holds all of the type and member functions common
to all the category of template classes.

The specialized traits for CHAR_T, 'char_traits<CHAR_T>' shall hold
all of the types and member functions defined in all of the categories
of the template classes(currently, those in clause 21 and clause 27).

Two specialized traits, 'char_traits<char>' and 'char_traits<wchar_t>'
shall be provided.

20.5.1 Definitions

[lib.utilities.traits.definitions]

Additional definitions of terms:

```
########################################################################
## In the following description, those of 'character' and 'character
## container type' are the same as in 27.1.1, so the corresponding
## items in 27.1.1 now should be removed.
########################################################################
```

-- character: In clause 21, 27, and subclause 20.5, the term "character" means any unit elements which, treated sequenatially, can represent text. The term does not only mean char and wchar_t type objects, but any value which can be repersented by a type which provides the definition specified in this clause.

-- character container type: Character container type is a class or a type used to represent a {\it character}. It is used for one of the template parameters of the string and iostream class templates. A character container class shall have a tribial constructor and destructor and a copy constructor and copu assignment operator that preserves its value and semantics.

-- traits: Traits is a class (structure) which represents a set of the defined types and functions neccesary for handling character objects in any implementation of the string library and/or the iostream library.

-- NTCTS (Null Terminated Character Type Sequence): A {\it null-terminated character type string} is a sequence of character type, charT, that precede the termination null character type value (charT(0)).

20.5.2 Character traits requirements   [lib.utilities.traits.requirements
]

```
namespace std {
  template <class charT> struct char_traits<charT> {
    typedef charT char_type;

    static void assign (char_type& c1, char_type c2);
    static bool eq (char_type c1, char_type c2);
    static bool ne (char_type c1, char_type c2);
    static bool lt (char_type c1, char_type c2);

    static int compare (const char_type* s1, const char_type* s2, size_t n);
    static size_t length (const char_type* s);
    static const char_type* find (const char_type* s, int n, const char_type& a);
    static char_type* copy (char_type* s1, const char_type* s2, size_t n);
    static char_type* assign (char_type* s, size_t n, char_type a);
  };
}
```

The template struct 'char_traits' takes a parameter which represents character container type, spelled 'charT' in the following, and holds a part of these types and static member functions that relate to the character container type, charT.

Those related to any other types specific to a library clause (for example, related to int_type, pos_type, off_type and state_type in iostreams) are described in each clause.

In the following subclauses (20.5.2.1, 20.5.2.2), the token ’charT’ represents the parameter of the traits.

20.5.2.1 defined type ’char_type’   [lib.utilities.traits.char.type]

```
    typedef charT char_type;
```

Description:

  The defined type, ’char_type’ is used to refer the template parameter of ’char_traits’ in the implementation of the string/iostream libraries.

Requires:

  charT shall provide constructor whose argument is an ’char’ type. Two values, charT(0), and charT(’\n’), represent end-of-string and newline character, respectively, for the character container type.

20.5.2.2 traits members

```
#######################################################################
# The following is based on the description of 27.1.1.2
#######################################################################
```

In each of the following function descriptions, paragraphs labelled by ’Related Libraries:’ specify the library category whose classes need the function so as to ensure their behavior.

static void assign (char_type& c1, char_type c2);

  Effects: Assigns c2 to c1.

  Related Libraries: string(clause 21) and, iostream(clause 27).

static bool eq (char_type c1, char_type c2);

  Returns: c1 == c2

  Related Libraries: string(clause 21) and, iostream(clause 27).

static bool ne (char_type c1, char_type c2);

  Returns: !c1 == c2

  Related Libraries: string(clause 21) and, iostream(clause 27).

static bool lt (char_type c1, char_type c2);

  Returns: c1 < c2.

  Related Libraries: string(clause 21).

static int compare (const char_type* s1, const char_type* s2, size_t n);

  Effects:

```
  for (size_t i = 0; i < n; ++i, ++s1, ++s2)
    if (ne (*s1, *s2))
      return lt (*s1, s2) ? -1 : 1;
  return 0;
```

Related Libraries: string(clause 21).

```
static size_t length (const char_type* s);
```

Effects: return the length of NTCTS pointed to by the parameter, s.

```
  size_t len = 0;
  while (ne (*s++, charT(0))) ++len;
  return len;
```

Returns: the length of NTCTP pointed to by the parameter, s.

Related Libraries: string(clause 21) and, iostream(clause 27).

```
static char_type* copy (char_type* s1, const char_type* s2, size_t n);
```

Effects:

```
  char_type *s = s1;
  for (size_t i = 0; i < n; ++i) assign (*s1++, *s2++);
  return s;
```

Related Libraries: string(clause 21) and, iostream(clause 27).

```
static const char_type* find (const char_type* s, int n, const char_type& a);
```

Effects: Determines the lowest pointer p, if possible, such that all of the following conditions hold true:

```
  + *p == a
  + s <= p < s + n
```

Returns: p if the function can determine such a value for p. Otherwise, returns 0.

Related Libraries: string(clause 21).

```
static char_type* move (char_type* s1, const char_type* s2, size_t n);
```

Effects: Copies elements. For each integer i in the range [0,n), performs assign (s1[i], s2[i]). Even when s2 is in the range [s1,s1+n), the implementation shall copy the character correctly.

Returns: s1.

Related Libraries: string(clause 21) and, iostream(clause 27).

```
static char_type* assign (char_type* s, size_t n, char_type a);
```

Effects: For each integer i in the range [0,n), performs assign(s[i],a).

  Returns: s.

Related Libraries: string(clause 21) and, iostream(clause 27).

20.5.2.4 char_traits specializations   [lib.utilities.char.traits.specializations]

```
namespace std{
  struct char_traits<char>;
  struct char_traits<wchar_t>;
}
```

The header <utilities> declares two structs that are specializations of the template struct ’char_traits’.

The struct char_traits<char> is the ’char’ type specialization of the template struct ’char_traits’, which contains all of the types and functions necessary to ensure the behaviors of the classes in clause 21 and 27.

The types and static member functions are described in 27.1.2[lib.iostreams.reqmts.char.traits] in detail.

21.1.1.2[lib.string.char.traits.members]
CHANGE ALL OF THE DESCRIPTION IN 21.1.1.2 WITH THE FOLLOWING;
##

21.1.1.2 char_traits members

Requirements:

Traits applied as one of the template parameters, traits, to some of the classes in this clause shall contain all of the types and the following types and functions specified in 20.5.2[lib.utilitles.traits.requirements].

```
   char_type;
   static void assign(char_type& c1, char_type c2);
   static bool eq(char_type c1, char_type c2);
   static bool ne(char_type c1, char_type c2);
   static bool lt(char_type c1, char_type c2);
   static int compare(const char_type* s1, const char_type* s2, size_t n);
   static size_t length(const char_type* s);
   static const char_type* find(const char_type* s, int n, const char_type& a);
   static char_type* copy(char_type* s1, const char_type* s2, size_t n);
   static char_type* assign char_type* s, size_t n, char_type a);
```

##
Replace all of the occurence ’string_char_traits’ with ’char_traits’
in the clause 21.
##

27.1.1 [lib.iostreams.definitions]
Remove descritpitopthe two items, "character" and "character
contaioner type" from this subclause because they moved to 20.5.1.
Append the following definition of "underlaid multibyte character
stream"

##

-- underlaid multibyte character stream: We sometimes use byte-stream medium(for example, files, or communication channels) in exchanging character sequense between other entities outside the program execution. In case we want to support such a medium by iostream, we represent a character object with a sequence of one or more number of bytes, so it can be regard as a multibyte character, such a byte-stream medium is called {\it underlaid multibyte character stream}.

In streambuf (or its derived) class object, we need to convert between a character represented in a character container type and a multibyte character. This conversion is performed to invoke member functions locale::codecnv facet with the imbued locale object. These member functions need a conversion state, represented with a 'state_type' object, related to the current point (byte offset) on the underlaid multibyte character stream, represented with a 'streamoff' object.

When repositioning occur in such a streambuf class object, it requires not only the current point but the related conversion state, so 'pos_type', which holds enough information to repositioning, shall have information equivalent to a pair of 'streamoff' and 'state_type'.

Implimantation may provide the wchar_t specialized version of 'basic_filebuf' class which treat byte stream on file as an multibyte character string. In this case, a type 'mbstate_t' defined in <cwchar>is available as 'state_type'.

27.1.2 [lib.iostreams.typeandmember.reqmts]
Rewrite all of the subclause as follows
##

27.1.2 requirements of char_traits members   [lib.iostreams.reqmts.char.traits]

The following describes a set of defined types and static member functions necessary for the iostream classes one of whose template parameter is traits. The traits provided in the specialization of the iostream classes shall holds all of these types and member functions.

```
  char_type;
  int_type;
  off_type;
  pos_type;
  state_type;
  static void assign (char_type& c1, char_type c2);
  static bool eq (char_type c1, char_type c2);
  static bool ne (char_type c1, char_type c2);
  static bool lt (char_type c1, char_type c2);
  static int compare (const char_type* s1, const char_type* s2, size_t n);
  static size_t length (const char_type* s);
  static char_type* copy (char_type* s1, const char_type* s2, size_t n);
  static const char_type* find (const char_type* s, int n, const char_type& a);
  static char_type* move (char_type* s1, const char_type* s2, size_t n);
  static char_type* assign (char_type* s, size_t n, char_type a);
  static char_type not_eof (int_type c);
  static char_type to_char_type (int_type c);
  static bool eq_int_type (int_type c1, int_type c2);
```

```
static state_type get_state (pos_type pos);
static pos_type get_pos (streampos fpos, state_type state);
```

27.1.2.1 defined type 'char_type'   [lib.iostreams.char.types]

```
typedef CHAR_T char_type;
```

The type, 'char_type' is used to refer the character container type in the implementation of the library classes defined in this clause.

Related Classes:

All of the classes that has a traits parameters defined in this clause.

27.1.2.3 defined type 'int_type'   [lib.iostreams.int.types]

```
typedef INT_T int_type;
```

Requires:

For a certain character container type, CHAR_T, a related container type, INT_T shall be a type or a class which can represents all of the valid characters converted from the corresponding CHAR_T type values as well as an end-of-file value, INT_T(EOF).

The type, 'int_type' represents another character container type which can holds end-of-file to be used as the return type of some of the iostream class member functions.

Related Classes:

All of the classes that has a traits parameters defined in this clause.

27.1.2.4 defined type 'off_type'   [lib.iostreams.off.types]

```
typedef OFF_T off_type;
```

Requires:

For a streambuf or its derived classes specialized for CHAR_T and TRAIT_T, a type or class, OFF_T is used to define 'off_type' in the traits and represent offsets to positional information(footnote 221) for the specialized streambuf (or its derived) class.

It is used to represent:

-- a signed displacement, measured in characters, from a specified position within a sequence.

-- an absolute position within a sequence.

The value OFF_T(-1) can e used as an error indicator.

The effect of passing to any function defined in clause 27 an OFF_T value not obtained from a function defined in clause 27 (for example, assigned an arbitrary integer), is undefined, except where otherwise noted.

Convertible to type POS_T.(Footnote 222). But no validity of the resulting POS_T value is ensured, whether or not the OFF_T value is valid.

Related Classes:

All of the classes that has a traits parameters defined in this clause.

27.1.2.5 defined type 'pos_type' [lib.iostreams.pos.types]

```
typedef POS_T pos_type;
```

Requires:

For a streambuf or its derived classes specialized for CHAR_T and TRAIT_T, a type or class, POS_T is used to define 'pos_type' for seek operation. It can store all the information necessary to reposition on the specialized streambuf (or its derived) class.

The type, 'pos_type' represent the type used for seek operation in the implementation of the streambuf (and its derived) classes defined in this clause.

Related Classes:

All of the classes that has a traits parameters defined in this clause.

27.1.2.6 defined type 'state_type' [lib.iostreams.state.type]

```
typedef STATE_T state_type;
```

Requires:

For a streambuf or its derived classes whose underlaid stream is multibyte character stream and specialized for CHAR_T and TRAIT_T, a type or class, STATE_T is used to define 'state_type' in the traits and represent the conversion state type or class which is applied to 'codecnv' facet defined in clause 22(localization library).

\##
\## The following is the same in 27.1.2.6, please cut & paste.
\##

In the following table,

- -- 'P' refers to type POS_T,
- -- 'p' and 'q' refer to an values of type POS_T,
- -- 'O' refers to type OFF_T,
- -- 'o' refers to a value of type OFF_T,
- -- 'i' refers to a value of type 'int'.

<<<<Table 77: should be inserted here>>>

The beavior of the stream after restoring the position with a POS_T value modified using any other arithmetic operation is undefined.

The stream operations whose return type is POS_T may return POS_T(OFF_T(-1)) as an invalid POS_T value to signal an error.

The conversion POS_T(OFF_T(-1)) constructs the invalid POS_T value, which is available only for comparing to the return value of such member functions.

Related Classes:

All of the streambuf or its derived classes whose underlaid stream is multibyte character stream.

27.1.2.8 Static member functions [lib.iostreams.traits.functions]

The following static member functions shall satisfy the requirements specified in the 20.5.2.2.

```
static void assign (char_type& c1, char_type c2);
static bool eq (char_type c1, char_type c2);
static bool ne (char_type c1, char_type c2);
static bool lt (char_type c1, char_type c2);
static int compare (const char_type* s1, const char_type* s2, size_t n);
static size_t length (const char_type* s);
static char_type* copy (char_type* s1, const char_type* s2, size_t n);
static const char_type* find (const char_type* s, int n, const char_type& a);
static char_type* move (char_type* s1, const char_type* s2, size_t n);
static char_type* assign (char_type* s, size_t n, char_type a);
```

As well as the above functions, the traits, TRAITS_T shall holds the following static member functions so as to ensure the behavior of classes specified in "Related Classes".

static char_type not_eof (int_type c);

Returns: c, if c != int_type(EOF), otherwise some value not 'int_type(EOF)'.

Related Classes: All of the classes that has a traits parameters defined in this clause.

static char_type to_char_type (int_type c);

Returns: c, if c != int_type(EOF), otherwise an unspecified value.

Related Classes: All of the classes that has a traits parameters defined in this clause.

static bool eq_int_type (int_type c1, int_type c2);

Returns: true if c1 is equal to c2, otherwise false.

Related Classes: All of the classes that has a traits parameters defined in this clause.

static state_type get_state (pos_type pos);

Returns: A 'state_type' value which represents the conversion state in the obhect 'pos'.

```
  Related Classes: All of the streambuf or its derived classes whose
    underlaid stream is multibyte character stream.


static pos_type get_pos (streampos fpos, state_type state);

  Effects: Constructs a 'pos_type' value which represent the stream
    position on the underlaid multibyte character stream specified by
    'fpos' and the conversion state at the corresponding position on
    the underlaid multibyte character stream.

  Returns: A 'pos_type' value which consists of the values of 'fpos'
    and 'state'.

  Related Classes: All of the streambuf or its derived classes whose
    underlaid stream is multibyte character stream.

-----------------------------------------------------------------
### 27.1.2 [lib.iostreams.type.rqmts] ################################
## 27.1.2 has shrinked (where only SZ_T is defined) and shifted to
## 27.1.3.
######################################################################

27.1.3 Type requirements                              [lib.iostreams.type.rqmts]

There are a type needed for implementing the iostream class templates.

27.1.3.1 Type SZ_T

A type that represents one of the signed basic integral types.  It is
used to represent the number of characters transferred in an I/O
operation, or the size of I/O buffers.

--------------------------------------------------------------
### 27.4 [lib.iostreams.base] ########################################
## removed the declaration of 'ios_traits<char>' and 'ios_traits<wchar_t>
## from the 'Header <ios> synopsis'
######################################################################

----------------------------
### 27.4.1 [lib.stream.types] ########################################
## removed whole of this subsection because the role of this description
## has moved to 27.1.2.
######################################################################

-------------------------------
### 27.4.2 [lib.ios.special.char.traits] #############################
## rewrite all of the description with the follows;
## Now only char_traits<char>, and char_traits<wchar_t> are provided.
######################################################################

27.4.2 specialization of char_traits

27.4.2.1 struct char_traits<char>

namespase std {
  struct char_traits<char> {
    typedef char char_type;
    typedef int  int_type;
    typedef streampos pos_type;
    typedef streamoff off_type;
```

```
    typedef mbstate_t state_type;

    static void assign (char_type& c1, char_type c2);
    static bool eq (char_type c1, char_type c2);
    static bool ne (char_type c1, char_type c2);
    static bool lt (char_type c1, char_type c2);
    static int compare (const char_type* s1, const char_type* s2, size_t n);
    static size_t length (const char_type* s);
    static const char_type* find (const char_type* s, int n, char_type a);
    static char_type* copy (char_type* s1, const char_type* s2, size_t n);
    static char_type* assign (char_type* s, size_t n, char_type a);

    static char_type not_eof (int_type c);
    static char_type to_char_type (int_type c);
    static bool eq_int_type (int_type c1, int_type c2);
    static state_type get_state (pos_type pos);
    static pos_type get_pos (streampos fpos, state_type state);
  };
}
```

As described 20.5.2.3, the header <utilities> declares a specialization of the template struct, 'char_traits<char>'. It is for narrow-oriented iostream classes. It holds all of the types and member functions described in 27.1.2 ([lib.iostreams.typeandmember.reqmts]).

The defined types for 'int_type', 'pos_type', 'off_type' and 'state_type' is 'int', 'streampos', 'streamoff', 'mbstate_t' respectively.

The type, 'streampos' is an implementation-defined type that satisfies the requirements in 27.1.2.5.

The type, 'streamoff' is an implementation-defined type that satisfied the requirements in 27.1.2.4.

The type, 'mbstate_t' is defined in <cwchar>, which can represent any of the conversion states possible to occur in implementation-defined set of supported multibyte character encoding rules.

27.4.2.2 struct char_traits<wchar_t>

```
namespace std {
  struct char_traits<wchar_t> {
    typedef wchar_t char_type;
    typedef wint_t  int_type;
    typedef wstreampos pos_type;
    typedef wstreamoff off_type;
    typedef mbstate_t state_type;

    static void assign (char_type& c1, char_type c2);
    static bool eq (char_type c1, char_type c2);
    static bool ne (char_type c1, char_type c2);
    static bool lt (char_type c1, char_type c2);
    static int compare (const char_type* s1, const char_type* s2, size_t n);
    static size_t length (const char_type* s);
    static const char_type* find (const char_type* s, int n, char_type a);
    static char_type* copy (char_type* s1, const char_type* s2, size_t n);
    static char_type* assign (char_type* s, size_t n, char_type a);

    static char_type not_eof (int_type c);
```

```
    static char_type to_char_type (int_type c);
    static bool eq_int_type (int_type c1, int_type c2);
    static state_type get_state (pos_type pos);
    static pos_type get_pos (streampos fpos, state_type state);
  };
}
```

As described 20.5.2.3, the header <utilities> declares a specialization of the template struct, 'char_traits<wchar_t>'. It is for wide-oriented iostream classes. It holds all of the types and member functions described in 27.1.2 ([lib.iostreams.typeandmember.reqmts]).

The defined types for 'int_type', 'pos_type', 'off_type' and 'state_type' is 'wint_t', 'wstreampos', 'wstreamoff', 'mbstate_t' respectively.

The type, 'wstreampos' is an implementation-defined type that satisfies the requirements in 27.1.2.5.

The type, 'wstreamoff' is an implementation-defined type that satisfied the requirements in 27.1.2.4.

Two pairs of types, 'streampos' and 'wstreampos', and 'streamoff' and 'wstreamoff' may be different, respectively in such implementation that adopt no shift encoding in narrow-oriented iostreams but supports one or more shift encodings in wide-oriented iostreams.

The type, 'mbstate_t' is defined in <cwchar>, which can represent any of the conversion states possible to occur in implementation-defined set of supported multibyte character encoding rules.

```
------------------------------------------------------------
######################################################################
## Replace all of the occurence 'ios_traits' with 'char_traits' in
## the clause 27.
######################################################################
```