Doc.No. J16/97-0013
        WG21/N1051
Project Prog. Lang. C++
Date   28 January 1997
Author Dag Bruck
        dag@dynasim.se

Removing error-prone default arguments from the library

[Nb. I submit these comments so they can be considered at the March meeting. They will probably resurface as NB comments on CD2.]

Instroduction

The C++ working paper uses default arguments in 32 places, mostly in the standard sequences deque, list and vector. There are reasons to consider removing some of them:

- default arguments effectively add an overloading, increasing the size of the library interface

- some functions taking default arguments are error-prone because their use is non-intuitive and suggest completely different operations.

Default arguments are used the standard library for:

- seq::seq creating "n" elements
- seq::assign(Size n, const T& t = T())
- seq::resize(size_type, T c = T())
- seq::insert(iterator, const T& t = T())
- complex::complex
- valarray::resize(size_t T c = T())
- basic_string::insert(iterator, charT)

where seq is deque, list, vector or vector<bool>.

I propose we increase library robustness by removing the default arguments for assign() and insert().

Discussion

Default arguments effectively specify overloadings. The standard library is big, and any reduction in size is beneficial to learning and using the library. In the case of assign() and insert(), so little utility is added by the default arguments that it does not warrant additional overloadings.

Taking advantage of the default argument for assign() and insert() leads to syntactic constructs that can easily be interpreted as something that is not intended by the standard. While the standard is clear in its specification, we must acknowledge that much programming is done without constant cross-checking with the specification. The current library is not user-robust; these examples are not unrealistic, and I could fall into similar traps myself.

Example: vector assign

```
      vector<int> v(20);
      v.assign(3);                  // 3 is my lucky number
```

This code does not assign 3 to each element of v as could be anticipated, instead it makes v a vector of three zero-valued elements. Also note that valarray has a proper element assignment:

```
      valarray va(20);
      va = 3;                       // 3 is my lucky number
```

Example: list insert

```
      list<int> a, b;
      a.insert(b.begin());
```

This code does not insert any element(s) of b into a. It is wrong because the first argument of insert() should be an iterator of a, but few implementations detect that.

Example: string insert

```
      string s;
      char* t = "Nellie";
      s.insert(t);
```

This is essentially the same example as above, assuming that string::iterator is char* (which is reasonable).

Proposal

Move that we amend the working paper by removing the default arguments of assign() and insert().

The affected document source lines (as of 10/25/96) would be:

```
lib-containers:
1081: void assign(Size n, const T& t = T());
1121: iterator insert(iterator position, const T& x = T());
1219: template <class Size, class T> void assign(Size n, const T& t = T());
1247: iterator insert(iterator position, const T& x = T());
1358: void assign(Size n, const T& t = T());
1395: iterator insert(iterator position, const T& x = T());
1489: template <class Size, class T> void assign(Size n, const T& t = T());
1515: iterator insert(iterator position, const T& x = T());
2041: template <class Size, class T> void assign(Size n, const T& t = T());
2081: iterator insert(iterator position, const T& x = T());
2166: template <class Size, class T> void assign(Size n, const T& t = T());
2230: iterator insert(iterator position, const T& x = T());
2338: template <class Size, class T> void assign(Size n, const T& t = T());
2378: iterator insert(iterator position, const bool& x = bool());

lib-strings:
925: iterator insert(iterator p, charT c = charT());
```

==
I have already sent Beman Dawes these minor issues, so I assume they are already on some issue list.

[lib.deque] [lib.list] [lib.vector] several places

The standard library sequences have a member declared

```
      template <class Size, class T>
      void assign(Size n, const T& t = T());
```

it is defined as

```
      erase(begin(), end());
      insert(begin(), n, t);
```

Maybe there is something profound I have completely missed, but I don't see why it is a member template function. Looking at the definition of insert(), it follows from Table 77 that

```
      n     must be a value of size_type
      t     must be a value of value_type (i.e., T)
```

From this follows that assign() could be declared as

```
      void assign(size_type n, const T& t = T());
```

[lib.sequence.reqmts] par 3

Function assign() is part of the standard sequences deque, list and vector, but not required from sequences in general (not listed in Table 77). Is it required or not?

[lib.string.capacity] par 3

What is "maximum size"? It can be deduced from reading about resize(), but I think it should be stated here.

[lib.string.capacity] par 8 and 9

Is reserve() guaranteed to accept any argument, even size_type(-1)? I think the description of capacity() is unclear, it doesn't stand for itself. Maybe we should define it as:

```
      Returns: a value not less than the value of res_arg of the
            last call of reserve(), or an unspecified value if
            reserve() has not been called for this object.  The
            returned value is not less than size().
```

or something along those lines.

[lib.string.insert] paragraph 10

The definition of basic_string::insert() does not include the default argument which is part of the synopsis in [lib.basic.string]. It probably should, as insert() of sequences have a default argument.

[lib.valarray.members] par 6

The example says that new elements are created using the default constructor. I think this is normative text that should be moved outside of the example.