Doc. No.: WG21/N0857=X3J16/96-0039
Date: 30 Jan 1996
Project: C++ Standard Library
Reply to: David Vandevoorde
vandevod@cs.rpi.edu

Assignment of valarrays

Description

The specifications of the valarray template (as currently included in the working paper) mandate non-conforming assignments to valarrays to cause the left-hand side to be resized; i.e., if a is a valarray of size 10 and b is a valarray of size 20, the expression

```
a = 2*b;
```

is required to change the size of a to 20.

Note that although:

```
valarray<int> a;
a += 2;
```

is allowed (adding two to every element of a), the following:

```
a = 2;
```

is not.

Finally, consider valarrays a and b of size N, and a valarray of integers P of the same size:

```
a = b[P];
```

is a straightforward permutation, but

```
a = a[P];
```

may correspond to a much more complex ‘‘in-place’’ permutation, which requires some temporary storage to do correctly. It was unclear to me if this is the intended behavior or if the result is undefined.

Discussion

The first issue (allowing non-conforming assignments) leads to:

1) loss of performance (only significant for very simple operations on small arrays or for architectures with very few registers),
2) a subtle difference between ‘a = a+b;’ and ‘a += b;’ (the former invalidates references to a, whereas the latter does not),
3) hard to find bugs.

The second issue (not allowing ‘a = 2;’) is minor, but nevertheless likely to lead to considerable frustration both in teaching and usage.

The third issue illustrated the more general issue of ‘‘indirect access order’’: does the DWP make guarantees as to the order in which elements of arrays are assigned. I believe it makes sense not to make such guarantees,

1) to avoid having to deal with the possibility of unexpected difficulties in the ‘‘simultaneous assignment’’ semantics

(the semantics that make ‘a = a[P]’ work),
2) to simplify the mapping of indirect access operations on architectures with specialized scatter/gather hardware.

Proposed Resolution

In [lib.valarray.members] 26.3.1.7, remove the fill member function and its annotations.

In [lib.valarray.access] 26.3.1.3/6, include regular assignments in the set of operations that do not invalidate references in a valarray.

Replace 26.3.1.2 by text along the lines of:

1 Assignment to an array does not invalidate references into that array.

```
template<typename T> valarray<T>& operator=(valarray<T> const&);
```

2 When invoked, this assignment operator causes each element of the *this array to be assigned the value of the corresponding element of the argument array. The behavior is undefined if these two arrays have different length.

3 Let $L denote the expression evaluating as *this and $R denote the expression evaluating as the argument of the operator (i.e., the right hand side). Let further P be an array whose elements are a permutation of the sequence [0, ..., this->size()-1].
If the effect of:

```
for (size_t i = 0; i<$L.size(); i++)
    $L[P[i]] = $R[i];
```

depends on the particular permutation that determines P, the result of the expression is undefined.

```
template<typename T> valarray<T>& operator=(T const&);
```

4 When invoked, this assignment operator causes each element of the *this array to be assigned the value of the argument.

5 Let $L denote the expression evaluating as *this and $R denote the expression evaluating as the argument of the operator (i.e., the right hand side). Let further P be an array whose elements are a permutation of the sequence [0, ..., this->size()-1].
If the effect of:

```
for (size_t i = 0; i<$L.size(); i++)
    $L[P[i]] = $R;
```

depends on the particular permutation that determines P, the result of the expression is undefined.

Note that little would change if the concept of a storage model were introduced (see another proposed resolution).