More About Valarrays

In addition to \texttt{valarray}s, there are four auxiliary types that specify subsets of a \texttt{valarray}: \texttt{slice_array}, \texttt{gslice_array}, \texttt{mask_array}, and \texttt{indirect_array}. We will briefly describe how each of them is used and the subsets of a \texttt{valarray} that they determine.

\textbf{SLICES.} One subset of a \texttt{valarray} is a \texttt{slice}, which selects every \textit{n}th element of a \texttt{valarray} for some integer \textit{n}. As we shall see, this in turn makes it possible to think of a \texttt{valarray} as a two-dimensional array having \textit{n} rows (or \textit{n} columns).

A declaration of a \texttt{slice} has the form

\texttt{slice s(start, size, stride);} 

which specifies the \texttt{size} indices \texttt{start}, \texttt{start + stride}, \texttt{start + 2*stride}, \ldots in a \texttt{valarray}. The member functions \texttt{start()}, \texttt{size()}, and \texttt{stride()} return the values \texttt{start}, \texttt{size}, and \texttt{stride}, respectively.

To illustrate their use, consider the \texttt{valarray} \texttt{v} and slices \texttt{s1}, \texttt{s2}, and \texttt{s3} defined by

\begin{verbatim}
  double d[] = {0,10,20,30,40,50,60,70,80,90,100,110};
  valarray<double> v(d, 12);
  slice s1(0,4,1), s2(4,4,1), s3(8,4,1);
\end{verbatim}

Then, \texttt{v[s1]}, \texttt{v[s2]}, and \texttt{v[s3]} are of type \texttt{slice_array} and contain the following values:

\begin{verbatim}
  v[s1]: 0, 10, 20, 30
  v[s2]:40, 50, 60, 70
  v[s3]:80, 90, 100, 110
\end{verbatim}

From this we see how these slices make it possible to view \texttt{v} as a $3 \times 4$ two-dimensional array:

\[

v = \begin{bmatrix}
0 & 10 & 20 & 30 \\
40 & 50 & 60 & 70 \\
80 & 90 & 100 & 110
\end{bmatrix}

\]

A \texttt{gslice} (generalized slice) contains essentially the information of \textit{n} slices; instead of one stride and one size, there are \textit{n} strides and \textit{n} sizes. The declarations of \texttt{gslice} objects are the same as for \texttt{slice}s, except that \texttt{size} and \texttt{stride} are \texttt{valarrays} whose elements are integer indices. To illustrate, consider the declarations
size_t sizearr[] = {2, 3}, stridearr[] = {4, 1};
valarray<
size_t> sz(sizearr, 2), str(stridearr, 2);
gslice gs(0, sz, str);

Then, v[gs] is of type gslice_array and contains: 0, 10, 20, 40, 50, 60. If we think of v as the preceding two-dimensional 3 × 4 array and gs as specifying that the size (sz) of the subarray to be selected is to be 2 × 3 and the strides (str) are to be 4 in the first dimension, 1 in the second, then v[gs] is the 2 × 3 subarray in the upper-left corner.

\[
v[gs] = \begin{bmatrix}
0 & 10 & 20 \\
40 & 50 & 60
\end{bmatrix}
\]

**Masks.** A mask_array provides another way to select a subset of a valarray. A mask is simply a boolean valarray, which when used as a subscript of a valarray, specifies for each index whether or not that element of the valarray is to be included in the subset.

To illustrate, consider the valarray v1 defined by

double d1[] = {0,10,20,30,40,50};
valarray<double> v1(d1, 6);

and the mask defined by

bool b[] = {true, false, false, true, true, false};
valarray<bool> mask(b, 6);

Then v2 and v3 defined by

valarray<double>
  v2 = v1[mask],       // 0, 30, 40
  v3 = pow(v1[mask], 2); // 0, 900, 1600

are of type mask_array and have the values indicated in the comments.

**Indirect Arrays.** An indirect_array specifies an arbitrary subset and reordering of a valarray. It is constructed by first defining a valarray of integers, which specify indices of the original valarray, where duplicate indices are allowed. For example, consider the valarray ind defined by
size_t indarr[] = {4, 2, 0, 5, 3, 1, 0, 5};
valarray< size_t > ind(indarr, 8);

Then valarray v4 defined by

valarray<double> v4 = v1[indarr];

is of type indirect_array and contains 40, 20, 0, 50, 30, 10, 0, 50.

EXERCISES

Exercises 1–4 deal with operations on n-dimensional vectors, which are sequences of n real numbers. In the description of each operation, A and B are assumed to be n-dimensional vectors:

\[ A = (a_1, a_2, \ldots, a_n) \]
\[ B = (b_1, b_2, \ldots, b_n) \]

Write functions for the operations, using valarrays to store the vectors. You should test your functions with driver programs.

1. Output an n-dimensional vector using <<.
2. Input an n-dimensional vector using >>.
3. Compute and return the magnitude of an n-dimensional vector:

\[ |A| = \sqrt{a_1^2 + a_2^2 + \ldots + a_n^2} \]

4. Compute and return the inner (or dot) product of two n-dimensional vectors (which is a scalar):

\[ A \cdot B = a_1*b_1 + a_2*b_2 + \ldots + a_n*b_n = \sum_{i=1}^{n} (a_i*b_i) \]