VALARRAYS

An important use of arrays is in vector processing and other numeric computation in science and engineering. In mathematics the term vector refers to a sequence (one-dimensional array) of real values on which various arithmetic operations are performed; for example, +, −, scalar multiplication, and dot product. Because much numeric work relies on the use of such vectors, highly-efficient libraries are essential in many fields. For this reason, C++ provides the standard library <valarray>, which is designed to carry out vector operations very efficiently.

Declarations of valarrays. A valarray declaration has one of the forms

\[
\text{valarray<T> V;}
\]
\[
\text{valarray<T> V(n);} \\
\text{valarray<T> V(value, n);} \\
\text{valarray<T> V(array);} \\
\text{valarray<T> V(w);} \\
\]

where \( T \) is a numeric type; \( n \) is an integer specifying the capacity of \( v \); \( value \) is a value of type \( T \); \( array \) is an array of \( T \) values; and \( w \) is a valarray.\(^1\) To illustrate, consider the following examples:

\[
\begin{align*}
\text{valarray<double> v0;} \\
\text{valarray<float> v1(100);} \\
\text{valarray<int> v2(999, 100);} \\
\text{const double a[] = {1.1, 2.2, 3.3, 4.4, 5.5};} \\
\text{valarray<double> v3(a, 4), v4(4, -1.0);} \\
\end{align*}
\]

The first declaration creates \( v0 \) as an empty valarray of doubles (which can be resized later); the second constructs \( v1 \) as a valarray containing 100 float values, initially 0; the third creates \( v2 \) as a valarray of 100 int values, initially 999; and the last creates \( v4 \) as a valarray of 4 doubles, initially the first four values \( 1.1, 2.2, 3.3, 4.4 \) stored in array \( a \), and \( v4 \) as a valarray of 4 doubles, initially \(-1.0\).

There are also four auxiliary types that specify subsets of a valarray: slice_array, gslice_array, mask_array, and indirect_array. These seem inappropriate, however, for a first course in computing and are thus left for the sequel\(^2\) to this text.

valarray Operations. The function members for valarrays are:

- the subscript operator \([\text{ ]}]\)

---

1. A valarray is actually a \textit{class template}. See Section 10.6 for more information about templates.

• assignment of same-size \texttt{valarrays}

• unary operations (applied elementwise): +, -, ~, !
  Example: \(-v3\) gives \(-1.1, -2.2, -3.3, -4.4\)

• assignment ops: +\,=, -=, *=, /=, %=, |=, ^=, <<=, >>=
  If * denotes one of these operations, \(v * x\) is equivalent to:
  \[
  \text{for (int } i = 0; i < v.\text{size()}; i++)}
  \]
  \[
  v[i] = v[i] * x;
  \]
  Example: \(v3 += v4\); changes \(v3\) to \(0.1, 1.2, 2.3, 3.4\)

• size(): the number of elements in the \texttt{valarray} (its capacity)
  Example: \(v3.\text{size()}\) is 4

• resize(n, val): reinitialize \texttt{valarray} to have \(n\) elements with (optional) value \(val\)
  Example:
  \[
  \text{cin >> n;}
  \]
  \[
  v0.\text{resize(n);}\]

• shift(n) and cshift(n): Shift values in the \texttt{valarray} \(|n|\) positions left if \(n > 0\), right
  if \(n < 0\). For \texttt{shift}, vacated positions are filled with 0; for \texttt{cshift}, values are shifted
circularly with values from the left end moving into the right end.
  Examples:
  \[
  v3.\text{shift(2); would change } v3 \text{ to } 3.3, 4.4, 0.0
  \]
  \[
  v3.\text{shift(-2); would change } v3 \text{ to } 0.0, 1.1, 2.2
  \]
  \[
  v3.\text{cshift(2); would change } v3 \text{ to } 3.3, 4.4, 1.1, 2.2
  \]

There also are several nonmember operations, which are applied elementwise:

• The following binary operators (applied elementwise):
  +, -, *, /, \%, \&, \|, \^, \langle\langle, \rangle\rangle, \&\&

  mathematical functions (from \texttt{cmath}): \texttt{atan2()}, \texttt{pow()}
  These operations and functions are applied elementwise. The operands may be \texttt{valarrays}
or a \texttt{valarray} and a scalar.

• The following mathematical functions, which are applied elementwise:
  \texttt{acos()}, \texttt{asin()}, \texttt{atan()}, \texttt{cos()}, \texttt{cosh()}, \texttt{exp()},
  \texttt{log()}, \texttt{log10()}, \texttt{sin()}, \texttt{sinh()}, \texttt{sqrt()}, \texttt{tan()}, \texttt{tanh()}

For example, the assignment statements
\[ v_4 = 2.0 \times v_3; \]
\[ w = \text{pow}(v_3, 2); \]

assign to \( v_4 \) the values 2.2, 4.4, 6.6, 8.8 and to \( w \) the squares of the elements of \( v_3 \), namely, 1.21, 4.84, 10.89, 19.36.

Some other operations that are useful with \texttt{valarrays} are found in the standard \texttt{<algorithm>} and \texttt{<numeric>} libraries (described in the Section 10.7 of the text). For example, \texttt{<numeric>} contains functions for calculating the sum of the elements in a sequence, the inner (dot) product of two sequences, the partial sums of a sequence, and differences of adjacent elements in a sequence.

\textbf{Input.} No predefined input operations are provided for \texttt{valarrays}, and so we must write our own input function to read values and store them in a \texttt{valarray} one at a time. The following code is an input function template. For maximum reusability, it receives the stream from which the values are to be extracted, so that the \texttt{valarray} can be input from the keyboard or from a file. Note that because a \texttt{valarray} carries its size (\texttt{size()}) along with it, there is no need to pass it as a parameter.

```cpp
/* read() fills a valarray<T> with input from a stream.  
* Note: Must #include <valarray> to use this function.  
* 
* Receives:  type parameter t  
*           in, an istream  
*           theValArray, a valarray  
* Input:    a sequence of T values  
* Precondition: operator >> is defined for type T.  
* Pass back: the modified istream and the  
*           modified valarray<T>  
******************************************************/

template <typename T>
void read(istream& in, valarray<T>& theValArray)
{
  for (int i = 0; i < theValArray.size(); i++)
    in >> theValArray[i];
}
```

\textbf{Output.} As with input, there is no output operation defined for \texttt{valarrays} and so a function to perform this operation must display the values in the \texttt{valarray} one at a time. Using a for loop like that in \texttt{read()} is the approach in the following function template \texttt{print().} Again note that because a \texttt{valarray} carries its size along with it, there is no need to pass it as a parameter.
/* print() displays the T values stored in a valarray.
* Note: Must #include <valarray> to use this function.
* 
* Receive: type parameter T
*           out, an ostream
*           theValArray, a valarray
* Output: each value in theArray to the ostream out
* Precondition: operator << is defined for type T.
* Passes back: the modified ostream out
***************************************************************************/

template <typename T>
void print(ostream& out, const valarray<T>& theValArray)
{
    for (int i = 0; i < theValArray.size(); i++)
        out << theValArray[i] << " ";
}