Vector classes

The files **g++-include/Vec.ccP** and **g++-include/AVec.ccP** provide pseudo-generic standard array-based vector operations. The corresponding header files are **g++-include/Vec.hP** and **g++-include/AVec.hP**. Class **Vec** provides operations suitable for any base class that includes an equality operator. Subclass **AVec** provides additional arithmetic operations suitable for base classes that include the full complement of arithmetic operators.

Vecs are constructed and assigned by copying. Thus, they should normally be passed by reference in applications programs.

Several mapping functions are provided that allow programmers to specify operations on vectors as a whole.

For illustrative purposes assume that classes **intVec** and **intAVec** have been generated via **genclass**.

Constructors and assignment

intVec a;	declares a to be an empty vector. Its size may be changed via resize.
intVec a(10);	declares a to be an uninitialized vector of ten elements (numbered 0-9).
intVec b(6, 0);	declares b to be a vector of six elements, all initialized to zero. Any value can be used as the initial fill argument.
a = b;	Copies b to a. a is resized to be the same as b.
a = b.at(2, 4)	constructs a from the 4 elements of b starting at b[2].

Assume declarations of **intVec a**, **b**, **c** and **int i**, **x** in the following.

Status and access

a.capacity();	returns the number of elements that can be held in a.
a.resize(20);	sets a's length to 20. All elements are unchanged, except that if the new size is smaller than the original, than trailing elements are deleted, and if greater, trailing elements are uninitialized.
a[i];	returns a reference to the i'th element of a, or produces an error if i is out of range.
a.elem(i)	returns a reference to the i'th element of a. Unlike the **[]** operator, i is not checked to ensure that it is within range.
a == b;	returns true if a and b contain the same elements in the same order.
a != b;	is the converse of a == b.

Constructive operations

c = concat(a, b);	sets c to the new vector constructed from all of the elements of a followed by all of b.

c = map(f, a);	sets c to the new vector constructed by applying int function f(int) to each element of a.
c = merge(a, b, f);	sets c to the new vector constructed by merging the elements of ordered vectors a and b using ordering (comparison) function f.
c = combine(f, a, b);	sets c to the new vector constructed by applying int function f(int, int) to successive pairs of a and b. The result has length the shorter of a and b.
c = reverse(a)	sets c to a, with elements in reverse order.

Destructive operations

a.reverse();	reverses a in-place.
a.sort(f)	sorts a in-place using comparison function f. The sorting method is a variation of the quicksort functions supplied with GNU emacs.
a.fill(0, 4, 2)	fills the 2 elements starting at a[4] with zero.

Other operations

a.apply(f)	applies function f to each element in a.

x = a.reduce(f, base)	accumulates the results of applying function f to successive elements of a starting with base.
a.index(int targ);	returns the index of the leftmost occurrence of the target, or -1, if it does not occur.
a.error(char* msg)	invokes the error handler. The default version prints the error message, then aborts.

AVec operations.

AVecs provide additional arithmetic operations. All vector-by-vector operators generate an error if the vectors are not the same length. The following operations are provided, for **AVecs a**, **b** and base element (scalar) **s**.

a = b;	Copies b to a. a and b must be the same size.
a = s;	fills all elements of a with the value s. a is not resized.
a + s; a - s; a * s; a / s	adds, subtracts, multiplies, or divides each element of a with the scalar.
a += s; a -= s; a *= s; a /= s;	adds, subtracts, multiplies, or divides the scalar into a.
a + b; a - b; product(a, b), quotient(a, b)	adds, subtracts, multiplies, or divides corresponding elements of a and b.
a += b; a -= b; a.product(b); a.quotient(b);	adds, subtracts, multiplies, or divides corresponding elements of b into a.

s = a * b;	returns the inner (dot) product of a and b.
x = a.sum();	returns the sum of elements of a.
x = a.sumsq();	returns the sum of squared elements of a.
x = a.min();	returns the minimum element of a.
x = a.max();	returns the maximum element of a.
i = a.min_index();	returns the index of the minimum element of a.
i = a.max_index();	returns the index of the maximum element of a.

Note that it is possible to apply vector versions other arithmetic operators via the mapping functions. For example, to set vector b to the cosines of doubleVec a, use **b = map(cos, a);**. This is often more efficient than performing the operations in an element-by-element fashion.