

Chapter 34 – Vectors*

* A number of compilers do not support vectors and other standard template classes at this writing.

1. The Standard C++ Vector Class

C++ comes with a default array class called *vector*. Objects of class *vector* are used like any other array, but also have several predefined timesaving operations. In addition, objects of class *vector* may contain an array of any native or programmer defined type or class.

2. Including Class Vector

Class *vector* is located in the standard header file *vector*, which must be included in your program.

3. Initializing Objects of Class Vector

There are two constructors in class *vector*. These have the following form:

```
vector <class-or-type> object(array-size);
vector <class-or-type> object(array-size, initial-value);
```

The following example declares a vector object named *weight* with an array element type of long and 1,000 elements.

```
vector <long> weight(1000);
```

The next example declares a vector object named *phonebook* with an array element type of the class *phonecard* and 100 elements.

```
vector <phonecard> phonebook(100);
```

This next example declares a vector object named *temperature* with an array element type of double and 10,000 elements. In *temperature*, each element is initialized to 0.0.

```
vector <double> temperature(10000, 0.0);
```

4. Accessing Elements in an Object of Class Vector

Elements contained in an object of class *vector* are accessed by treating the vector object as the name of the array. For example:

```
vector <double> temperature(24, 0.0);
for (int x=0; x<24; x++) {
    cout << "Enter a temperature: ";
    cin >> temperature[x];
}
```

If the array type is a user-defined class, all member functions are available with each element in the usual manner. The following example assumes that class `phonecard` has the non-void member function `getLastName()`:

```
vector <phonecard> phonebook(100);
    :
    :
for (int x=0; x<currentlength; x++)
    cout << phonebook[x].getLastName() << endl;
```

5. Vector Member Functions Capacity and Resize

The vector member functions *capacity* and *resize* work together to allow a vector object to have its array resized. *Capacity* is a non-void function that returns the maximum number of elements that the array in the vector object contains or can contain. *Resize* is a void function that takes as its only argument the new number of elements in the array of the vector object. Here is an example of using these functions:

```
vector <long> list(100);
int i=0;
infile >> list[i];
while (!infile.eof( )) {
    i++;
    if (i >= list.capacity( )) list.resize(list.capacity( ) + 100);
    infile >> list[i];
}
```

Programming Exercise 34.1

Rewrite the *phonebook* class used in Chapter 29.3 so that it is constructed with an array of the standard C++ vector class. Be sure and take advantage of the ease of dynamically expanding arrays of the standard C++ vector class.