Sentinel Loops

If an array contains a special value marking the end of the data, then you can use a **sentinel loop** to process the array. In the long-ago data-processing days, for instance, an array of positive values would contain a `-1` as a terminator or sentinel.

One **advantage** of the sentinel technique, is that it allows you to write functions that process variable amounts of data. (Later, we'll look at **partially-filled arrays**, which are a more general solution to this problem.)

Here's an example using a sentinel loop, which calculates the average grade on a test:

```
double average(const double grades[])
{
   double sum = 0;
   int count = 0;
   for (size_t i = 0; grades[i] >= 0; ++i)
   {
      sum += grades[i];
      count++;
   }
   return sum / count;
}
```

The **disadvantage of this technique** is that your function needs to trust that the array has been properly terminated, which is really a security hole. This technique is not used that widely for numeric arrays. However, C-style strings use a special sentinel (called the NUL byte) to mark the end of the string, and all C-string functions employ sentinel loops.