Then, the while test condition checks to see if there still is room left in the array (i < ArSize) and if the input value is nonnegative (temp >= 0). If so, it copies the temp value to the array and increases the array index by 1. At this point, because array numbering starts at 0, i equals the total number of entries to date. That is, if i starts out at 0, the first cycle through the loop assigns a value to naaaq[0] and then sets i to 1.

The loop terminates when the array is filled or when the user enters a negative number. Note that the loop reads another value into temp only if i is less than ArSize, that is, only if there still is room left in the array.

After it gets data, the program uses an if else statement to comment if no data were entered (that is, if the first entry was a negative number) and to process the data if any is present.

**Setting Up Ranges with &&**

The && operator also lets you set up a series of if else if else statements with each choice corresponding to a particular range of values. Listing 6.6 illustrates the approach. It also shows a useful technique for handling a series of messages. Just as a pointer-to-char variable can identify a single string by pointing to its beginning, an array of pointers-to-char can identify a series of strings. Simply assign the address of each string to a different array element. Listing 6.6 uses the qualify array to hold the addresses of four strings. For example, qualify[1] holds the address of the string "mud tug-of-war\n". The program then can use qualify[1] like any other pointer to a string—for example, with cout or with strlen() or strcmp(). Using the const qualifier protects these strings from accidental alterations.

**Listing 6.6 more_and.cpp**

// more_and.cpp -- use logical AND operator
#include <iostream>
using namespace std;
const char * qualify[4] =       // an array of pointers
{                               // to strings
  "10,000-meter race.\n",
  "mud tug-of-war.\n",
  "mud tug-of-war.\n",
  "mud tug-of-war.\n"};