Checkpoint

5.6 Name the three expressions that appear inside the parentheses in the `for` loop's header.

5.7 You want to write a `for` loop that displays "I love to program" 50 times. Assume that you will use a counter variable named `count`.

A) What initialization expression will you use?

B) What test expression will you use?

C) What update expression will you use?

D) Write the loop.

5.8 What will the following program segments display?

A)
```
for (int count = 0; count < 6; count++)
    cout << (count + count);
```

B)
```
for (int value = -5; value < 5; value++)
    cout << value;
```

C)
```
int x;
for (x = 5; x <= 14; x += 3)
    cout << x << endl;
cout << x << endl;
```

5.9 Write a `for` loop that displays your name 10 times.

5.10 Write a `for` loop that displays all of the odd numbers, 1 through 49.

5.11 Write a `for` loop that displays every fifth number, zero through 100.

5.7 Keeping a Running Total

CONCEPT: **A *running total* is a sum of numbers that accumulates with each iteration of a loop. The variable used to keep the running total is called an *accumulator*.**

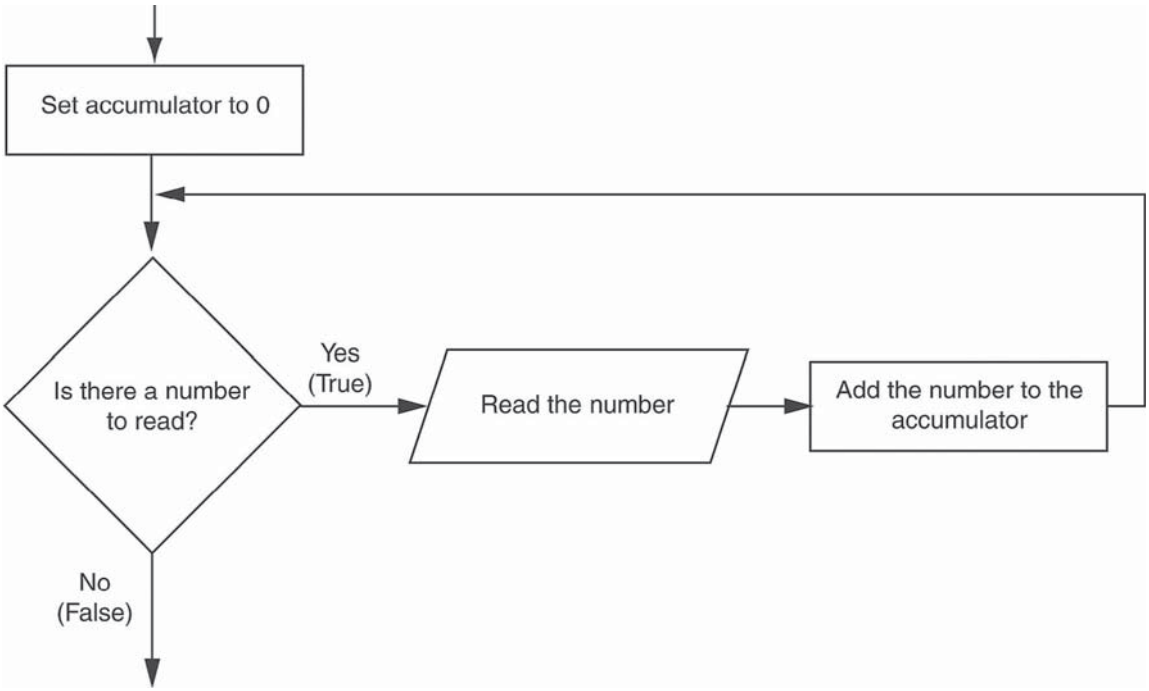
Many programming tasks require you to calculate the total of a series of numbers. For example, suppose you are writing a program that calculates a business's total sales for a week. The program would read the sales for each day as input and calculate the total of those numbers.

Programs that calculate the total of a series of numbers typically use two elements:

- A loop that reads each number in the series.
- A variable that accumulates the total of the numbers as they are read.

The variable that is used to accumulate the total of the numbers is called an *accumulator*. It is often said that the loop keeps a *running total* because it accumulates the total as it reads each number in the series. Figure 5-11 shows the general logic of a loop that calculates a running total.

Figure 5-11 Logic for calculating a running total



When the loop finishes, the accumulator will contain the total of the numbers that were read by the loop. Notice that the first step in the flowchart is to set the accumulator variable to 0. This is a critical step. Each time the loop reads a number, it adds it to the accumulator. If the accumulator starts with any value other than 0, it will not contain the correct total when the loop finishes.

Let's look at a program that calculates a running total. Program 5-12 calculates a company's total sales over a period of time by taking daily sales figures as input and calculating a running total of them as they are gathered.

Program 5-12

```
1  // This program takes daily sales figures over a period of time
2  // and calculates their total.
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  int main()
8  {
9       int days;           // Number of days
10      double total = 0.0; // Accumulator, initialized with 0
11
12      // Get the number of days.
13      cout << "For how many days do you have sales figures? ";
```

```
14      cin >> days;
15
16      // Get the sales for each day and accumulate a total.
17      for (int count = 1; count <= days; count++)
18      {
19          double sales;
20          cout << "Enter the sales for day " << count << ": ";
21          cin >> sales;
22          total += sales;  // Accumulate the running total.
23      }
24
25      // Display the total sales.
26      cout << fixed << showpoint << setprecision(2);
27      cout << "The total sales are $" << total << endl;
28      return 0;
29  }
```

Program Output with Example Input Shown in Bold

```
For how many days do you have sales figures? 5 [Enter]
Enter the sales for day 1: 489.32 [Enter]
Enter the sales for day 2: 421.65 [Enter]
Enter the sales for day 3: 497.89 [Enter]
Enter the sales for day 4: 532.37 [Enter]
Enter the sales for day 5: 506.92 [Enter]
The total sales are $2448.15
```

Let's take a closer look at this program. Line 9 defines the `days` variable, which will hold the number of days that we have sales figures for. Line 10 defines the `total` variable, which will hold the total sales. Because `total` is an accumulator, it is initialized with 0.0.

In line 14 the user enters the number of days that he or she has sales figures for. The number is assigned to the `days` variable. Next, the `for` loop in lines 17 through 23 executes. In the loop's initialization expression, in line 17, the variable `count` is defined and initialized with 1. The test expression specifies the loop will repeat as long as `count` is less than or equal to `days`. The update expression increments `count` by one at the end of each loop iteration.

Line 19 defines a variable named `sales`. Because the variable is defined in the body of the loop, its scope is limited to the loop. During each loop iteration, the user enters the amount of sales for a specific day, which is assigned to the `sales` variable. This is done in line 21. Then, in line 22 the value of `sales` is added to the existing value in the `total` variable. (Note that line 22 does *not* assign `sales` to `total`, but *adds* `sales` to `total`. Put another way, this line increases `total` by the amount in `sales`.)

Because `total` was initially assigned 0.0, after the first iteration of the loop, `total` will be set to the same value as `sales`. After each subsequent iteration, `total` will be increased by the amount in `sales`. After the loop has finished, `total` will contain the total of all the daily sales figures entered. Now it should be clear why we assigned 0.0 to `total` before the loop executed. If `total` started at any other value, the total would be incorrect.