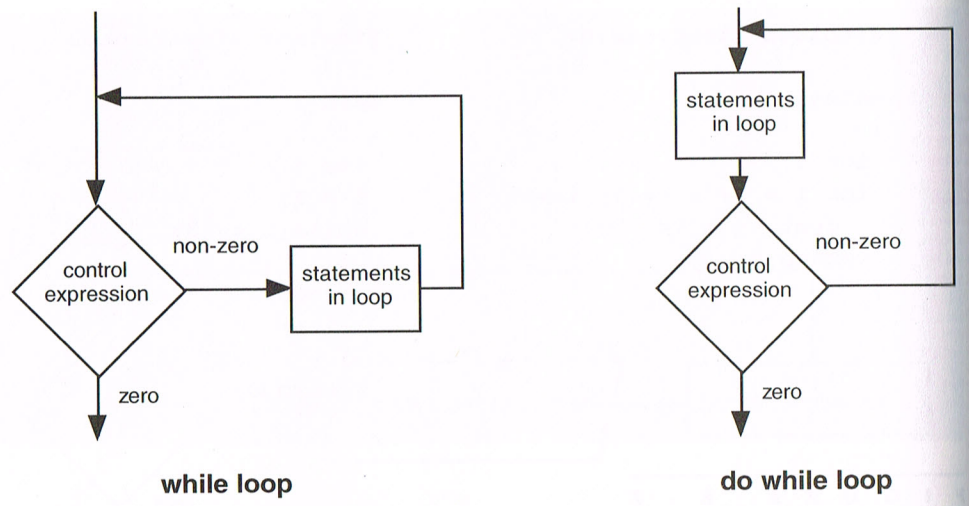

FIGURE 8-9
The difference between a while loop and a do while loop is where the control expression is tested.

On the Net

Modern operating systems often require that programs be ***event driven,*** *meaning that events such as the click of a mouse or a menu selection drives the flow of the program. At the heart of an event-driven program is a loop called an* ***event loop*** *which constantly iterates waiting for an event to occur. The program then takes action based on the event which has occurred. This differs from programs which pause at a prompt and wait for a command to be entered. To learn more about event-driven programming and event loops, see http://www.ProgramCPP.com. See topic 8.2.1.*

STOPPING IN THE MIDDLE OF A LOOP

The keyword *break*, also utilized with switch statements, can be used to end a loop before the conditions of the control expression are met. Once a break terminates a loop, the execution begins with the first statement following the loop. In the program you ran in Exercise 8-6, entering zero caused the program to end. But the program squares zero before it ends, even though the step is unnecessary. The program in Figure 8-10 uses a break statement to correct the problem.

In the program in Figure 8-10, the value entered by the user is tested with an if statement as soon as it is input. If the value is zero, the break statement is executed to end the loop. If the value is any number other than zero, the loop continues. The control expression can remain `num != 0` without affecting the function of the program. In this case, however, the break statement will stop the loop before the control expression is reached. Therefore, the control expression can be changed to 1 to create an infinite loop. The 1 creates an infinite loop because the loop continues to iterate as long as the control expression is true, which is represented by the value 1. The loop will repeat until the break statement is executed.

Note

You should allow the control expression to end an iteration structure whenever practical. Whenever you are tempted to use a break statement to exit a loop, make sure that using the break statement is the best way to end the loop.


```
#include <iostream.h>

main()
{
 float num, squared;
 do
  {
   cout << "Enter a number (Enter 0 to quit): ";
   cin >> num;
   if (num == 0)
     { break; }
   squared = num * num;
   cout << num << " squared is " << squared << '\n';
  }
 while (1);
 return 0;
}
```

FIGURE 8-10
The break statement ends the loop as soon as the value of zero is input.

The continue statement is another way to stop a loop from completing each statement. But instead of continuing with the first statement after the loop, the continue statement skips the remainder of a loop and starts the next iteration of the loop. Figure 8-11 shows an example of how the continue statement can be used to cause a for loop to skip an iteration.

The continue statement in Figure 8-11 causes the statements in the for loop to be skipped when the counter variable is 5. The continue statement also can be used in while and do while statements.

```
#include <iostream.h>

main()
{
 int i;
 for(i = 1; i <= 10; i++)
  {
   if (i == 5)
    continue;
   cout << i << '\n';
  }
 return 0;
}
```

FIGURE 8-11
The continue statement causes the number 5 to be skipped.

EXERCISE 8-7 USING THE continue STATEMENT

1. Open *CONTINUE.CPP.*
2. Compile and run the program. Notice that the number 5 does not appear in the output because of the continue statement.
3. Close the source file.