cout << "Factorial numbers that are <= " << bound << ":\n1, 1";  
long f=1;  
for (int i=2; f <= bound; i++)  
{ f *= i;  
cout << ", " << f;  
}  

Enter a positive integer: 1000000  
Factorial numbers < 1000000:  
1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880  

This for loop program has the same effect as the do..while loop program because it executes the same instructions. After initializing f to 1, both programs initialize i to 2 and then repeat the following five instructions: print f, multiply f by i, increment i, check the condition (f <= bound), and terminate the loop if the condition is false.  

The for statement is quite flexible, as the following examples demonstrate.  

EXAMPLE 4.13 Using a Descending for Loop  

This program prints the first ten positive integers in reverse order:  

```cpp  
int main()  
{ for (int i=10; i > 0; i--)  
cout << " " << i;  
}  
10 9 8 7 6 5 4 3 2 1  
```

EXAMPLE 4.14 Using a for Loop with a Step Greater than One  

This program determines whether an input number is prime:  

```cpp  
int main()  
{ long n;  
cout << "Enter a positive integer: ";  
cin >> n;  
if (n < 2) cout << n << " is not prime." << endl;  
else if (n < 4) cout << n << " is prime." << endl;  
else if (n%2 == 0) cout << n << " = 2*" << n/2 << endl;  
else  
{ for (int d=3; d <= n/2; d += 2)  
if (n%d == 0)  
{ cout << n << " = " << d << "*" << n/d << endl;  
exit(0);  
}  
cout << n << " is prime." << endl;  
}  

Enter a positive integer: 101  
101 is prime.  

Enter a positive integer: 975313579  
975313579 = 17*57371387  
```

Note that this for loop uses an increment of 2 on its control variable i.
EXAMPLE 4.15 Using a Sentinel to Control a for Loop

This program finds the maximum of a sequence of input numbers:

```cpp
int main()
{
    int n, max;
    cout << "Enter positive integers (0 to quit): ";
    cin >> n;
    for (max = n; n > 0; )
    {
        if (n > max) max = n;
        cin >> n;
    }
    cout << "max = " << max << endl;
}
```

This for loop is controlled by the input variable `n`; it continues until `n ≤ 0`. When an input variable controls a loop this way, it is called a sentinel.

Note the control mechanism `(max = n; n > 0; )` in this for loop. Its update part is missing, and its initialization `max = n` has no declaration. The variable `max` has to be declared before the for loop because it is used outside of its block, in the last output statement in the program.

EXAMPLE 4.16 Using a Loop Invariant to Prove that a for Loop is Correct

This program finds the minimum of a sequence of input numbers. It is similar to the program in Example 4.15:

```cpp
int main()
{
    int n, min;
    cout << "Enter positive integers (0 to quit): ";
    cin >> n;
    for (min = n; n > 0; )
    {
        if (n < min) min = n;
        // INVARIANT: min <= n for all n, and min equals one of the n
        cin >> n;
    }
    cout << "min = " << min << endl;
}
```

The full-line comment inside the block of the for loop is called a loop invariant. It states a condition that has two characteristic properties: (1) it is true at that point on every iteration of the loop; (2) the fact that it is true when the loop terminates proves that the loop performs correctly. In this case, the condition `min <= n for all n, and min equals one of the n` is always true because the preceding if statement resets the value of `min` if the last input value of `n` was less than the previous value of `min`. And the condition that `min equals one of the n` is always true because `min` is initialized to the first `n` and the only place where `min` changes its value is when it is assigned to a new input value of `n`. Finally, the fact that the condition is true when the loop terminates means that `min` is the minimum of all the input numbers. And that outcome is precisely the objective of the for loop.