for Loop

- Alternative counter loop
  - Could be done with a while loop

```
i = 0;
while( i < 5 )
{
    cout << i << " ";
    i++;
}
```

  - Provides a distinct, clear way to do counter loops

```
for( i = 0 ; i < 5 ; i ++ )
{
    cout << i << " ";
}
```

`for` Loop

```
for( i = 0 ; i < 5 ; i++ )
{
    cout << i << " ";
}
```

- Initialization

  `i = 0`

  – Sets the initial value of the counter variable

- Condition

  `i < 5`

  – Specifies the condition for continuing to loop

- Update

  `i++`

  – Updates the counter variable

`for` Loop

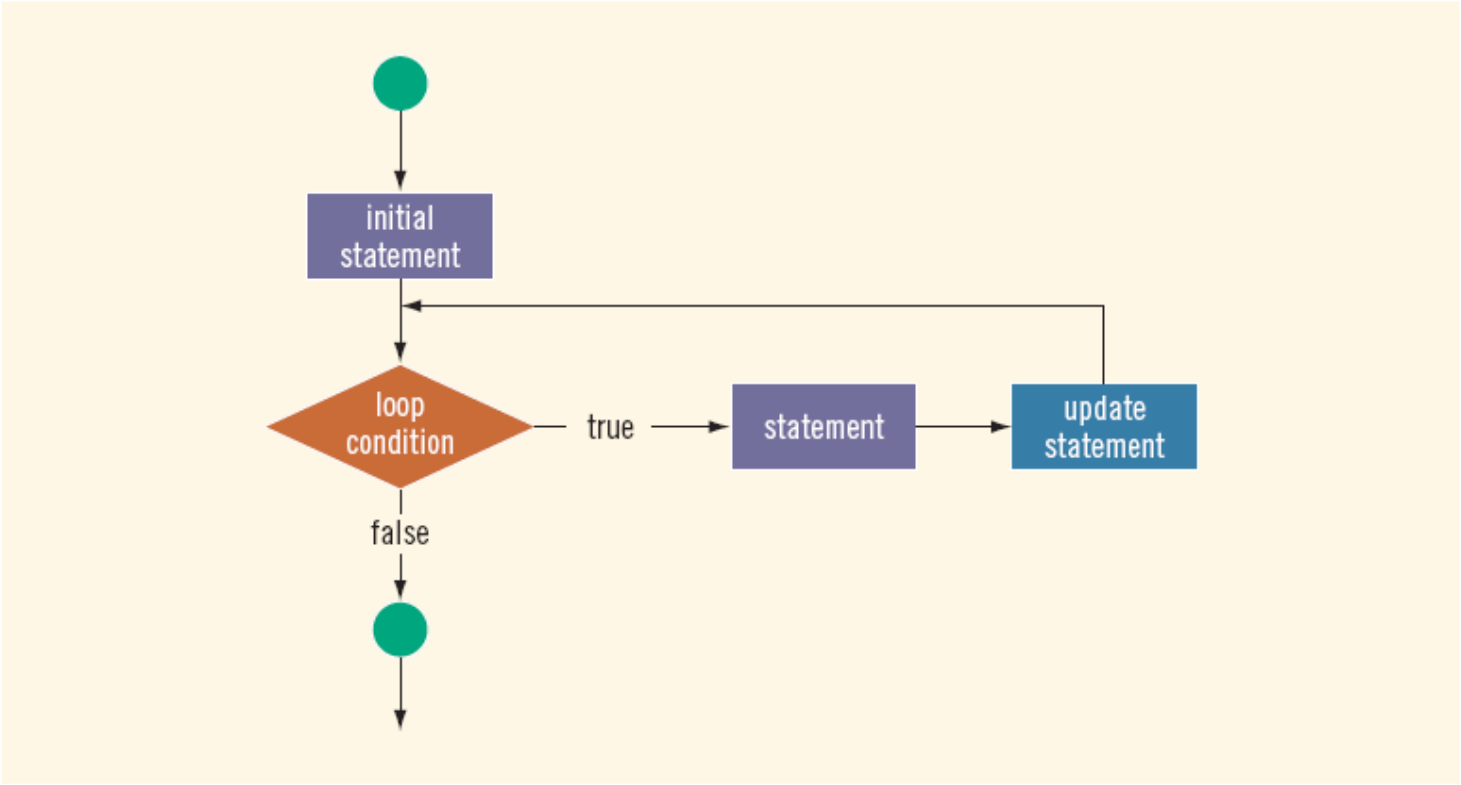

FIGURE 5-2 `for` loop

`for` Loop Examples

```
for( i = 0 ; i < 5 ; i++ )
{
    cout << i << " ";
}
```

- Different starting points
- Complex conditions
- Different counter updates
  - Increment (`i++`) vs. decrement (`i--`)
  - Count by multiples (`i = i + 3`)

`for` Loop Examples

EXAMPLE 5-7

The following `for` loop prints the first 10 non negative integers:

```
for (i = 0; i < 10; i++)
    cout << i << " ";
cout << endl;
```

EXAMPLE 5-8

1. The following `for` loop outputs `Hello!` and a star (on separate lines) five times:

```
for (i = 1; i <= 5; i++)
{
    cout << "Hello!" << endl;
    cout << "*" << endl;
}
```

2. Consider the following `for` loop:

```
for (i = 1; i <= 5; i++)
    cout << "Hello!" << endl;
    cout << "*" << endl;
```

`for` Loop Examples

EXAMPLE 5-10

You can count backward using a `for` loop if the `for` loop control expressions are set correctly.

For example, consider the following `for` loop:

```
for (i = 10; i >= 1; i--)
    cout << " " << i;
cout << endl;
```

The output is:

```
10 9 8 7 6 5 4 3 2 1
```

EXAMPLE 5-11

You can increment (or decrement) the loop control variable by any fixed number. In the following `for` loop, the variable is initialized to `1`; at the end of the `for` loop, `i` is incremented by 2. This `for` loop outputs the first `10` positive odd integers.

```
for (i = 1; i <= 20; i = i + 2)
    cout << " " << i;
cout << endl;
```

Choosing the Right Looping Structure

- All three loops have their place in C++
  - If you know or can determine in advance the number of repetitions needed, the `for` loop is the correct choice
  - If you do not know and cannot determine in advance the number of repetitions needed, and it could be zero, use a `while` loop
  - If you do not know and cannot determine in advance the number of repetitions needed, and it is at least one, use a `do...while` loop

Nested Loops

```
for (i = 1; i <= 5; i++)
{
    for (j = 1; j <= i; j++)
    {
        cout << "*";
    }
    cout << endl;
}
```

i	j

Nested Loops

```
for (i = 1; i <= 5; i++)
{
    for (j = 1; j <= i; j++)
    {
        cout << "*";
    }
    cout << endl;
}
```

- Outputs:

```
*
**
***
****
*****
```

i	j
1	1
2	1
	2
3	1
	2
	3
4	1
	2
	3
	4
5	1
	2
	3
	4
	5

Exercise

- Draw the pattern created by this code (the top `for` statement has changed):

```
for (i = 5; i >= 1; i--)
{
    for (j = 1; j <= i; j++)
    {
          cout << "*";
    }
    cout << endl;
}
```