Lecture Five
Loops and Repetition Statements

Loops and Repetition

- Loops in programs allow us to repeat blocks of code.
- Useful for:
  - Trying again for correct input
  - Counting
  - Repetitive activities
  - Programs that never end

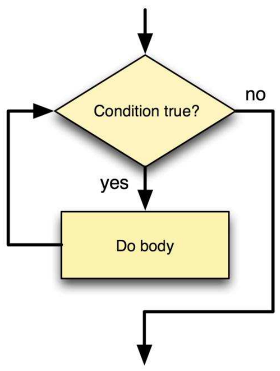
Three Types of Loops/Repetition in C

- while
  - top-tested loop (pre-test)
- for
  - counting loop
  - forever-sentinel
- do
  - bottom-tested loop (post-test)

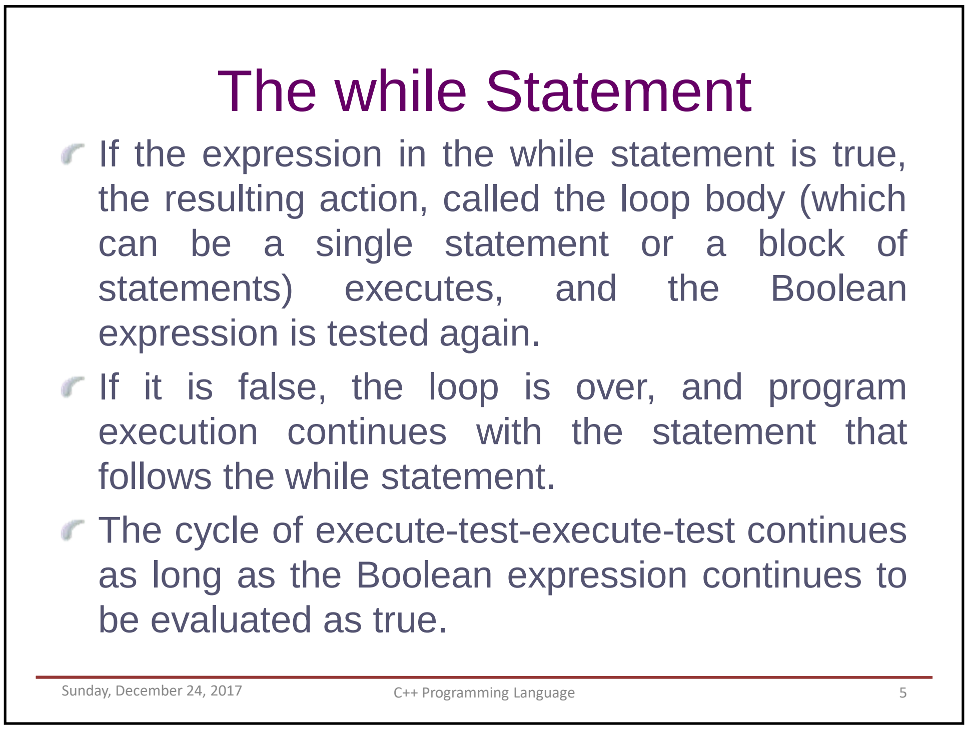
The while Statement

- An executing program checks the condition of a while statement before executing any of the statements in its body.
- The format of the while loop is as follows:

while (Boolean expression)
  action while expression is true;

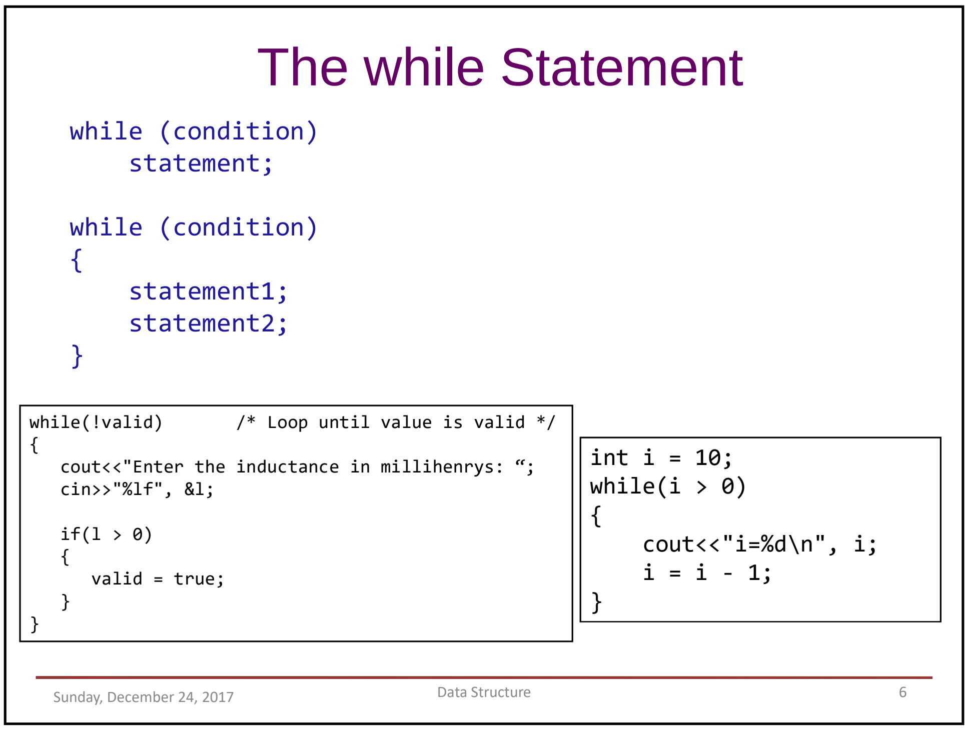


`while` flowchart



The while Statement

- If the expression in the while statement is true, the resulting action, called the loop body (which can be a single statement or a block of statements) executes, and the Boolean expression is tested again.
- If it is false, the loop is over, and program execution continues with the statement that follows the while statement.
- The cycle of execute-test-execute-test continues as long as the Boolean expression continues to be evaluated as true.



The while Statement

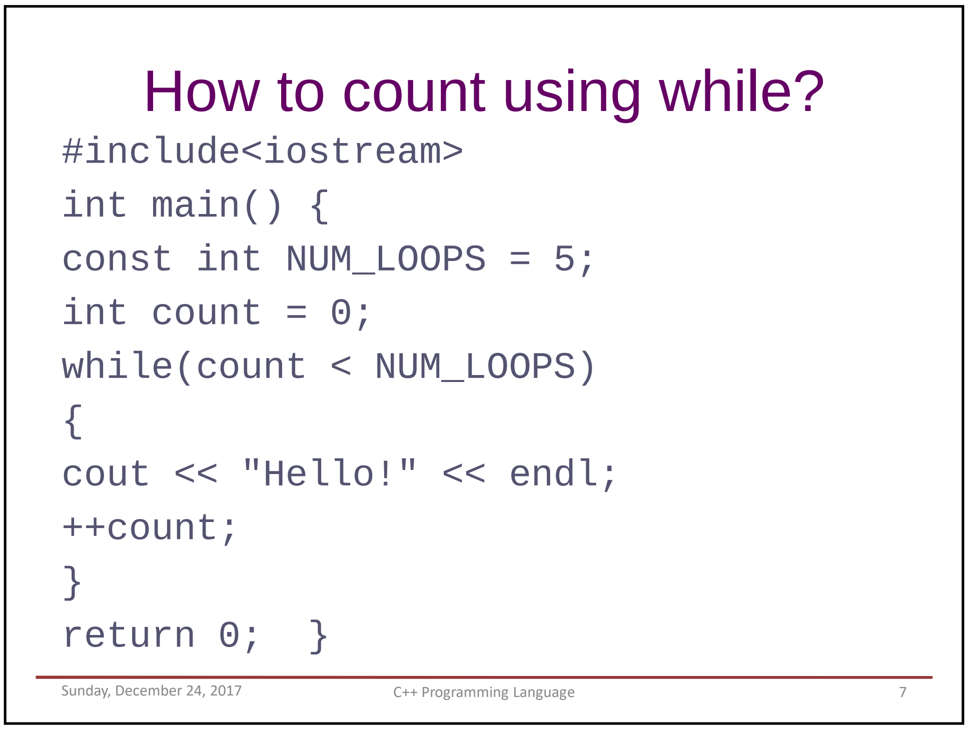
```
while (condition)
    statement;

while (condition)
{
    statement1;
    statement2;
}
```

```
while(!valid)       /* Loop until value is valid */
{
   cout<<"Enter the inductance in millihenrys: “;
   cin>>"%lf", &l;

   if(l > 0)
   {
      valid = true;
   }
}
```

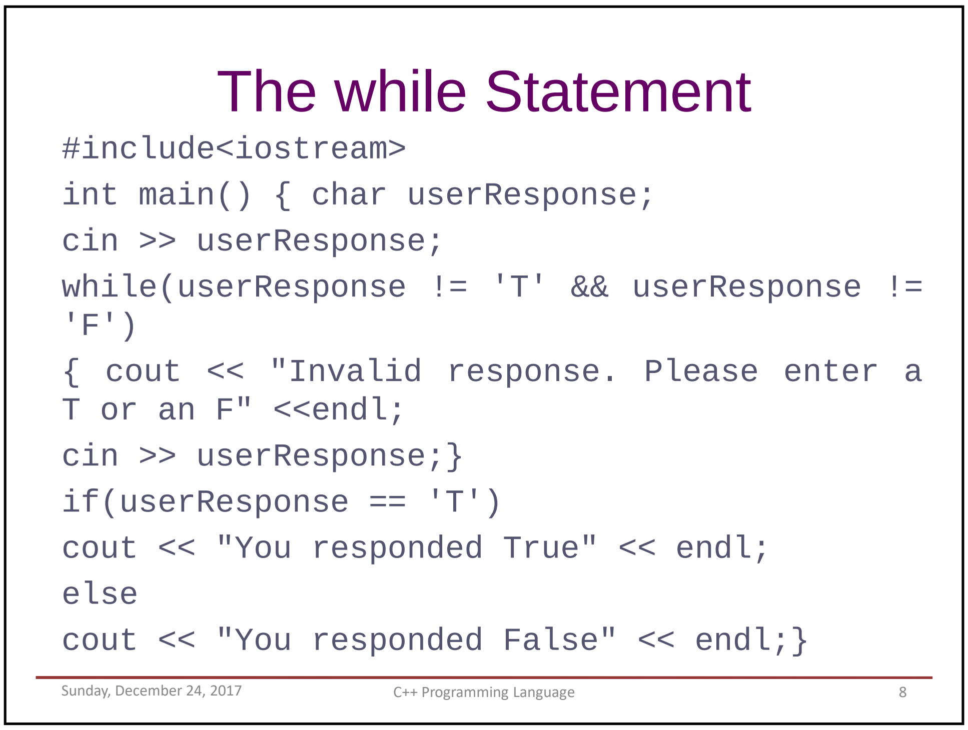
```
int i = 10;
while(i > 0)
{
    cout<<"i=%d\n", i;
    i = i - 1;
}
```



How to count using while?

```
#include<iostream>
int main() {
const int NUM_LOOPS = 5;
int count = 0;
while(count < NUM_LOOPS)
{
cout << "Hello!" << endl;
++count;
}
return 0;  }
```

Sunday, December 24, 2017 C++ Programming Language 7



The while Statement

```
#include<iostream>
int main() { char userResponse;
cin >> userResponse;
while(userResponse != 'T' && userResponse !=
'F')
{ cout << "Invalid response. Please enter a
T or an F" <<endl;
cin >> userResponse;}
if(userResponse == 'T')
cout << "You responded True" << endl;
else
cout << "You responded False" << endl;}
```

Sunday, December 24, 2017 C++ Programming Language 8

The while Statement

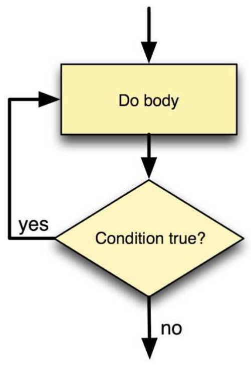
```
number = 1;
while(number <= 10)
{
cout << number << endl;
++number;
}
```

```
Number=1;
while(number<=10)
  cout << number << endl;
++number;
```

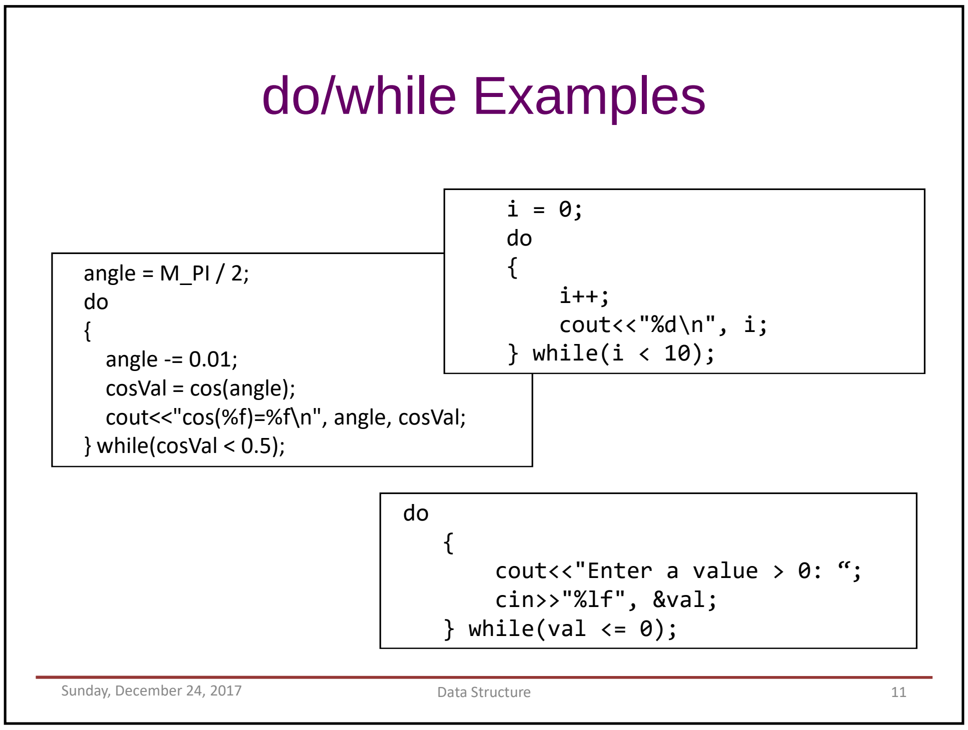
The do while Statement

- Sometimes this sequence of checking the condition first then executing the body is inconvenient.

```
include <iostream.h>
int main() {
int in_value = -1;
cout << "Please enter an integer in the range 0-10: ";
while (in_value < 0 || in_value > 10)
    cin >> in_value;
cout << "Legal value entered was " << in_value << '\n';
}
```



do/while flowchart



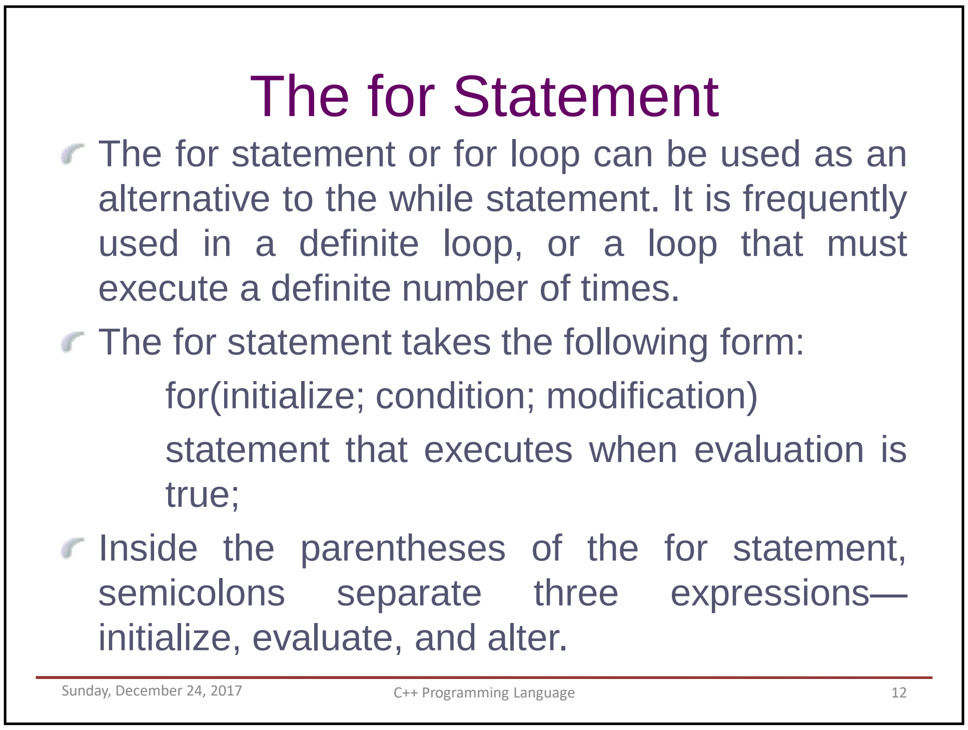
do/while Examples

```
i = 0;
do
{
    i++;
    cout<<"%d\n", i;
} while(i < 10);
```

```
angle = M_PI / 2;
do
{
   angle -= 0.01;
   cosVal = cos(angle);
   cout<<"cos(%f)=%f\n", angle, cosVal;
} while(cosVal < 0.5);
```

```
do
   {
        cout<<"Enter a value > 0: “;
        cin>>"%lf", &val;
   } while(val <= 0);
```

Sunday, December 24, 2017 Data Structure 11



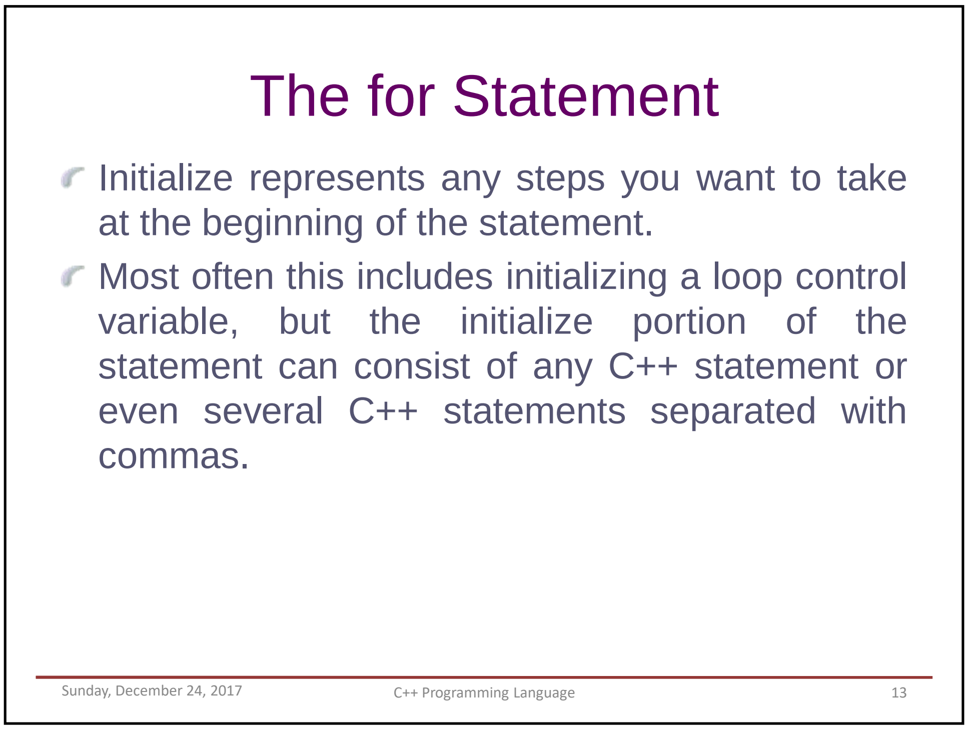
The for Statement

- The for statement or for loop can be used as an alternative to the while statement. It is frequently used in a definite loop, or a loop that must execute a definite number of times.
- The for statement takes the following form:

  for(initialize; condition; modification)

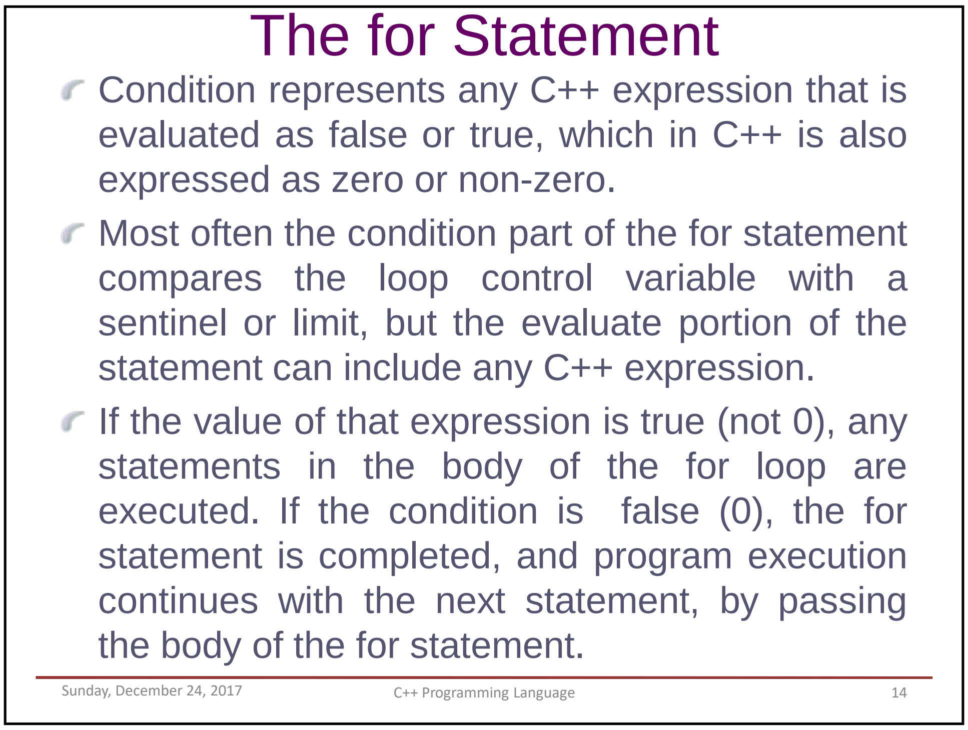
  statement that executes when evaluation is true;
- Inside the parentheses of the for statement, semicolons separate three expressions—initialize, evaluate, and alter.

Sunday, December 24, 2017 C++ Programming Language 12



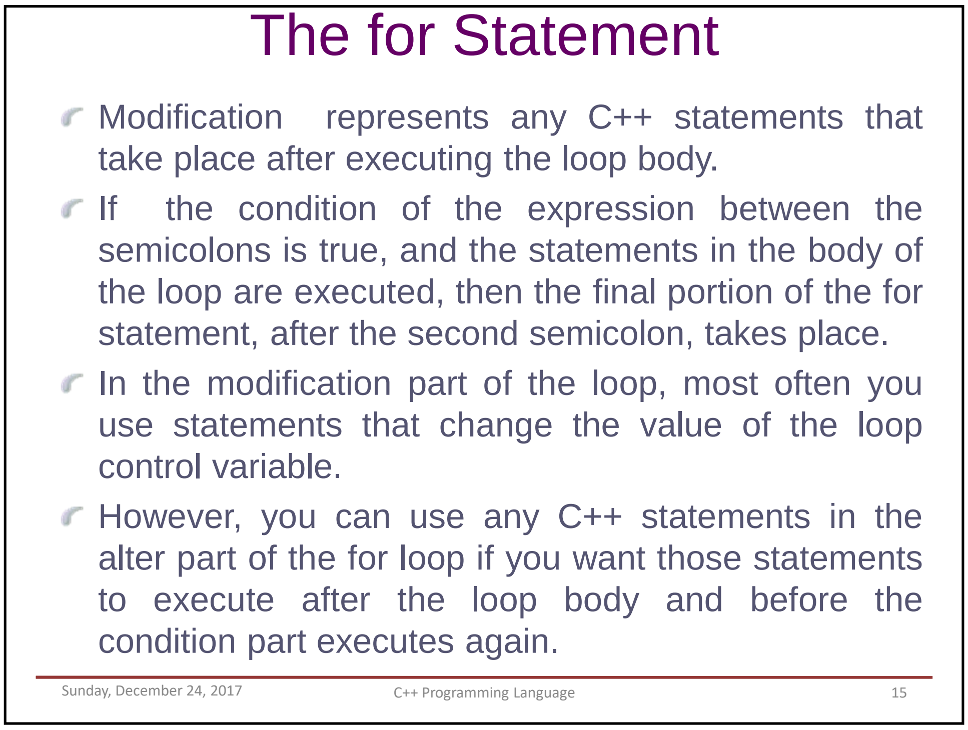
The for Statement

- Initialize represents any steps you want to take at the beginning of the statement.
- Most often this includes initializing a loop control variable, but the initialize portion of the statement can consist of any C++ statement or even several C++ statements separated with commas.



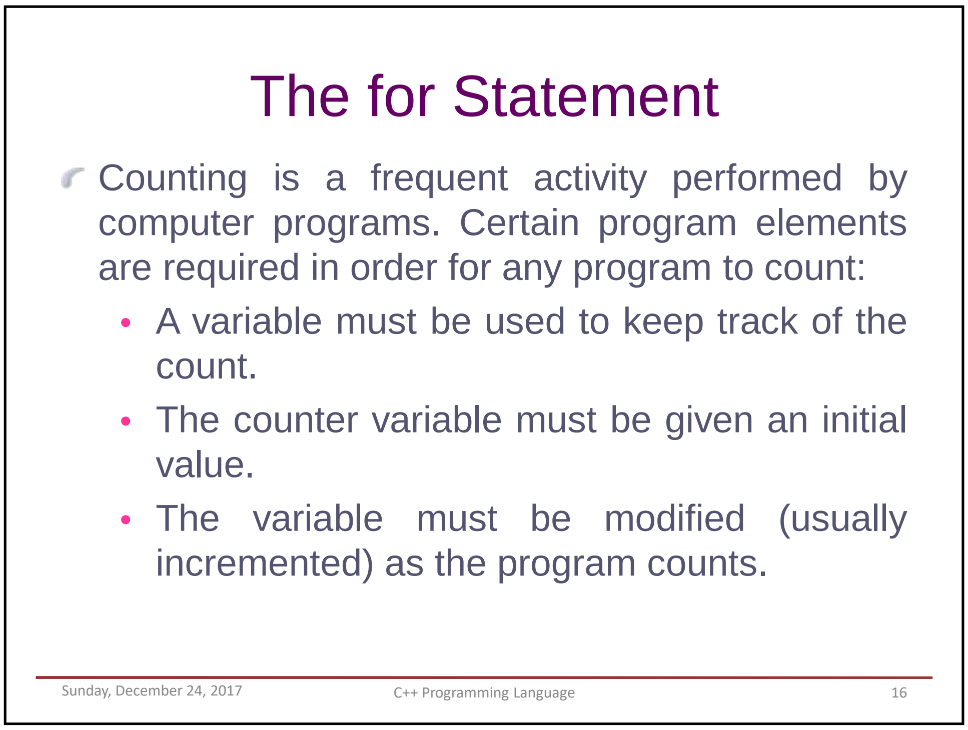
The for Statement

- Condition represents any C++ expression that is evaluated as false or true, which in C++ is also expressed as zero or non-zero.
- Most often the condition part of the for statement compares the loop control variable with a sentinel or limit, but the evaluate portion of the statement can include any C++ expression.
- If the value of that expression is true (not 0), any statements in the body of the for loop are executed. If the condition is false (0), the for statement is completed, and program execution continues with the next statement, by passing the body of the for statement.



The for Statement

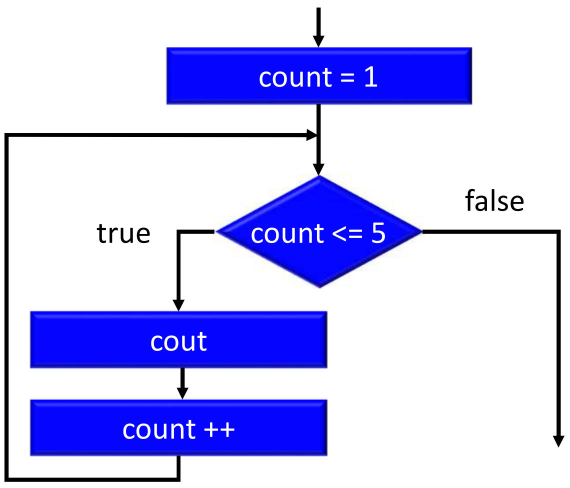
- Modification represents any C++ statements that take place after executing the loop body.
- If the condition of the expression between the semicolons is true, and the statements in the body of the loop are executed, then the final portion of the for statement, after the second semicolon, takes place.
- In the modification part of the loop, most often you use statements that change the value of the loop control variable.
- However, you can use any C++ statements in the alter part of the for loop if you want those statements to execute after the loop body and before the condition part executes again.



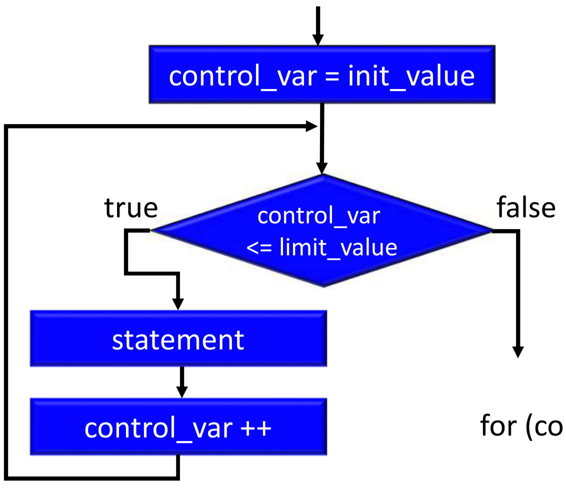
The for Statement

- Counting is a frequent activity performed by computer programs. Certain program elements are required in order for any program to count:
  - A variable must be used to keep track of the count.
  - The counter variable must be given an initial value.
  - The variable must be modified (usually incremented) as the program counts.

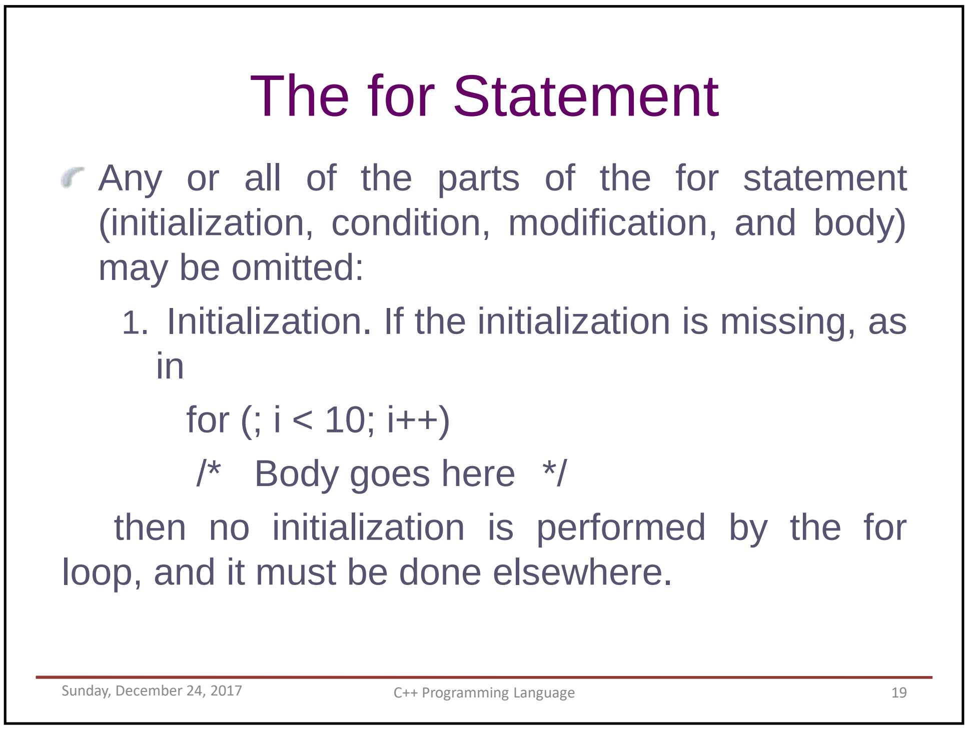
for(count=1; count<=5; count++) cout<<“count=%d\n”, count;



Ascending for <=,++



```
for (control_var=init_value;
        control_var <=limit_value;
        control_var++) statement;
```



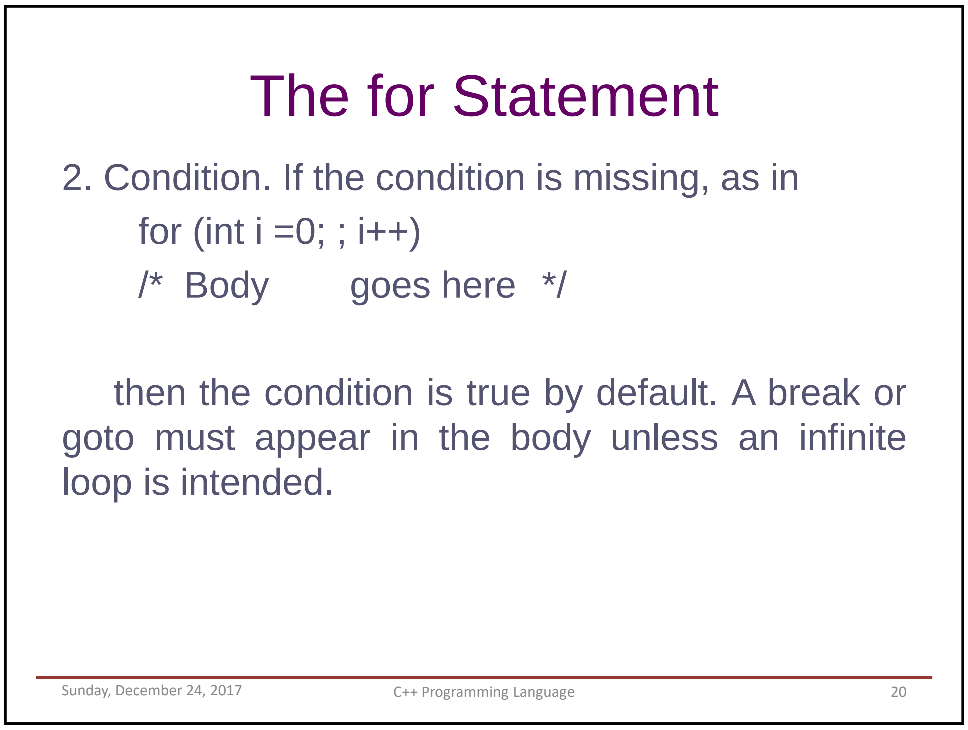
The for Statement

- Any or all of the parts of the for statement (initialization, condition, modification, and body) may be omitted:
  1. Initialization. If the initialization is missing, as in

     for (; i < 10; i++)

     /* Body goes here */

  then no initialization is performed by the for loop, and it must be done elsewhere.



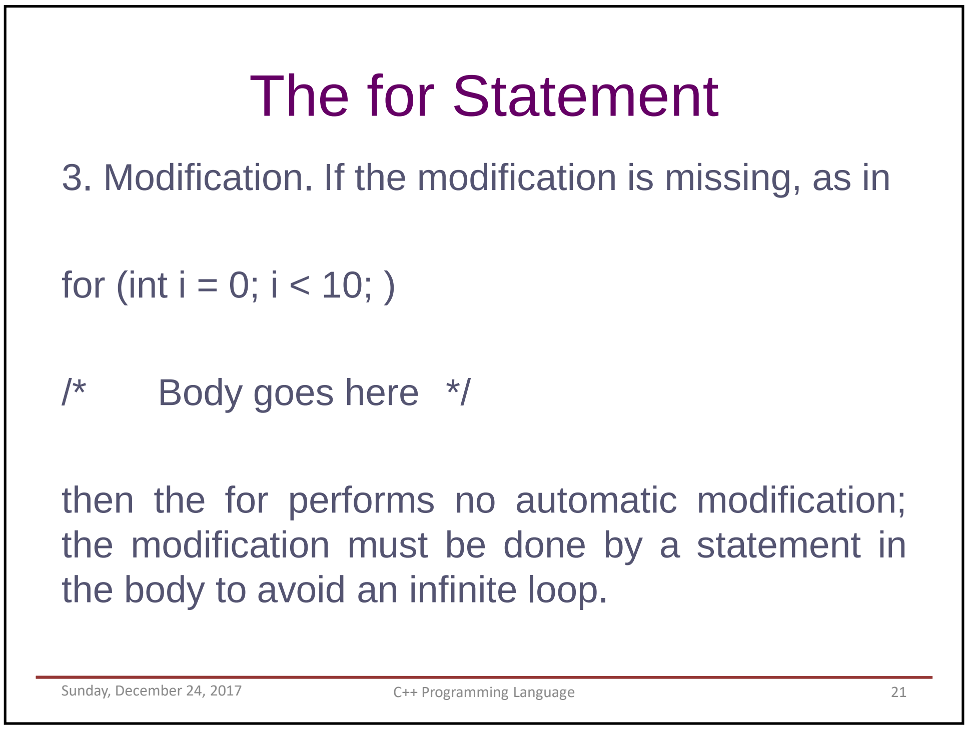
The for Statement

2. Condition. If the condition is missing, as in

   for (int i =0; ; i++)

   /* Body goes here */

then the condition is true by default. A break or goto must appear in the body unless an infinite loop is intended.



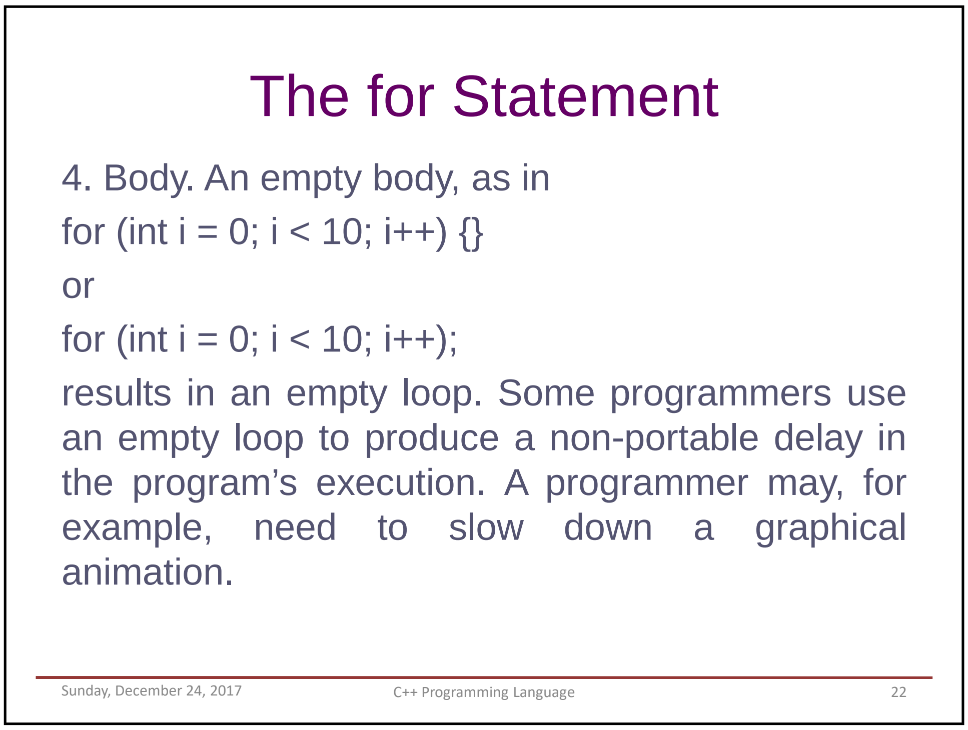
The for Statement

3. Modification. If the modification is missing, as in

```
for (int i = 0; i < 10; )

/*      Body goes here   */
```

then the for performs no automatic modification; the modification must be done by a statement in the body to avoid an infinite loop.



The for Statement

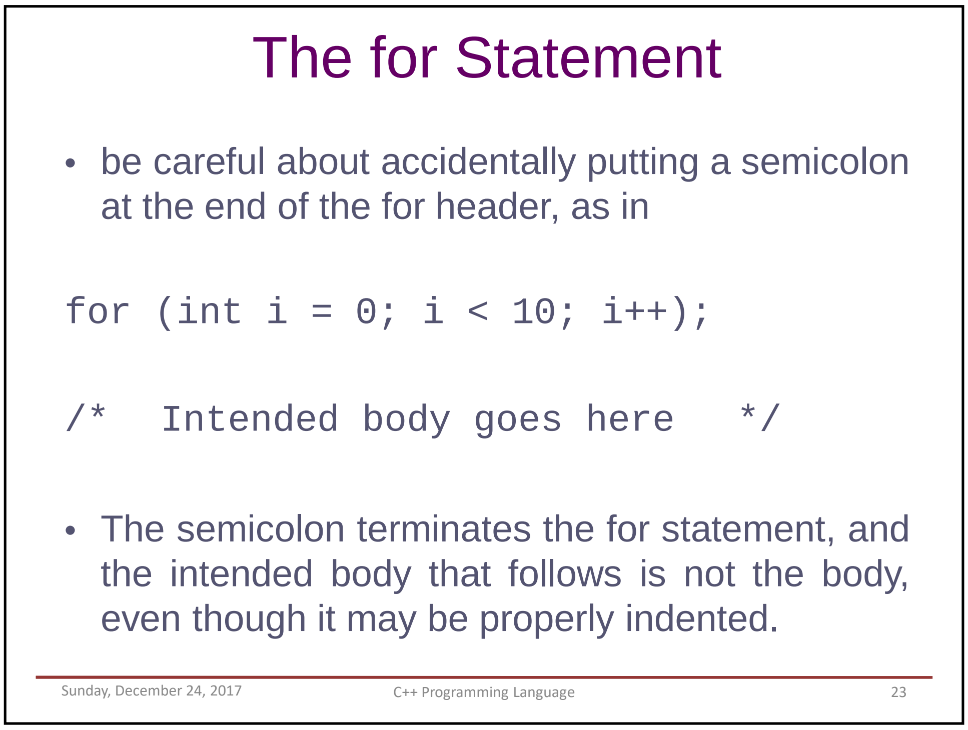
4. Body. An empty body, as in

```
for (int i = 0; i < 10; i++) {}
```

or

```
for (int i = 0; i < 10; i++);
```

results in an empty loop. Some programmers use an empty loop to produce a non-portable delay in the program’s execution. A programmer may, for example, need to slow down a graphical animation.



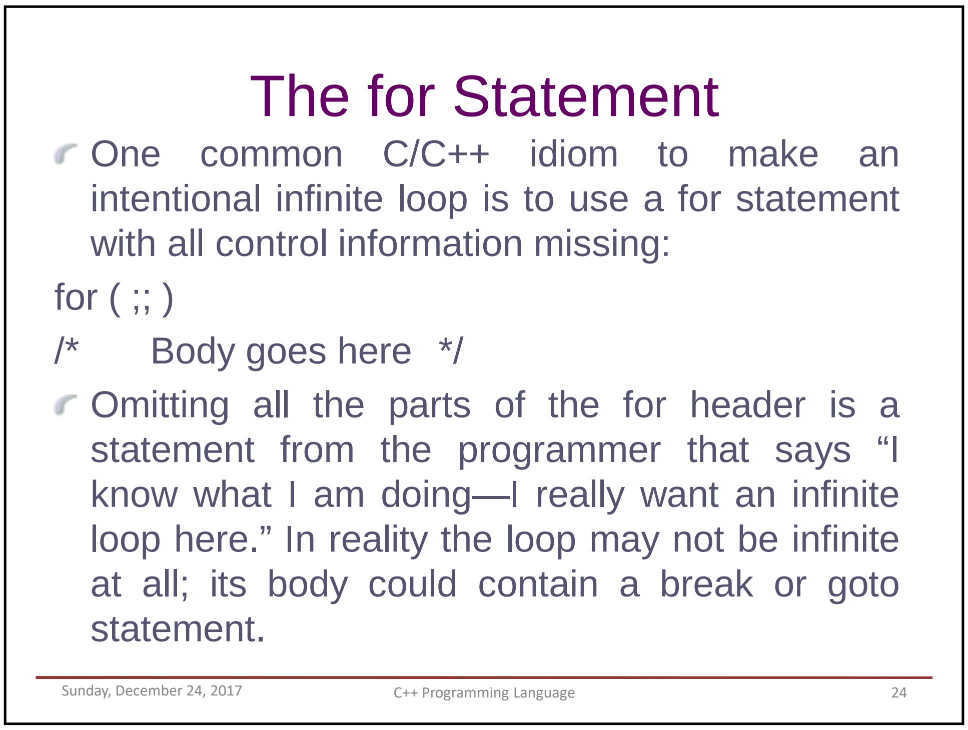
The for Statement

- be careful about accidentally putting a semicolon at the end of the for header, as in

```
for (int i = 0; i < 10; i++);

/*   Intended body goes here   */
```

- The semicolon terminates the for statement, and the intended body that follows is not the body, even though it may be properly indented.

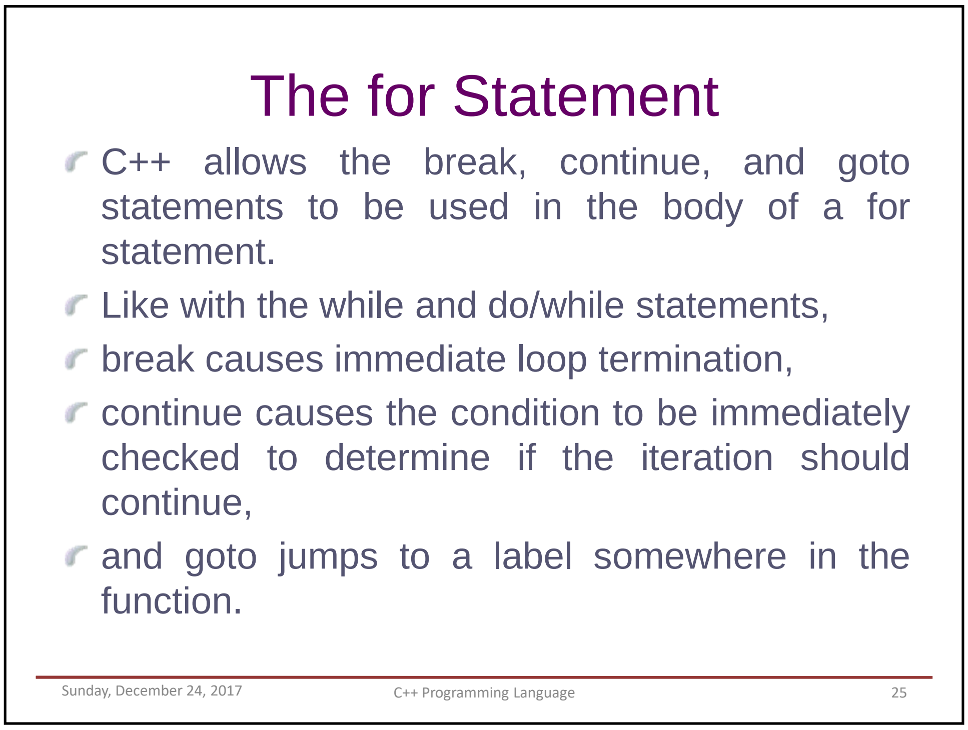


The for Statement

- One common C/C++ idiom to make an intentional infinite loop is to use a for statement with all control information missing:

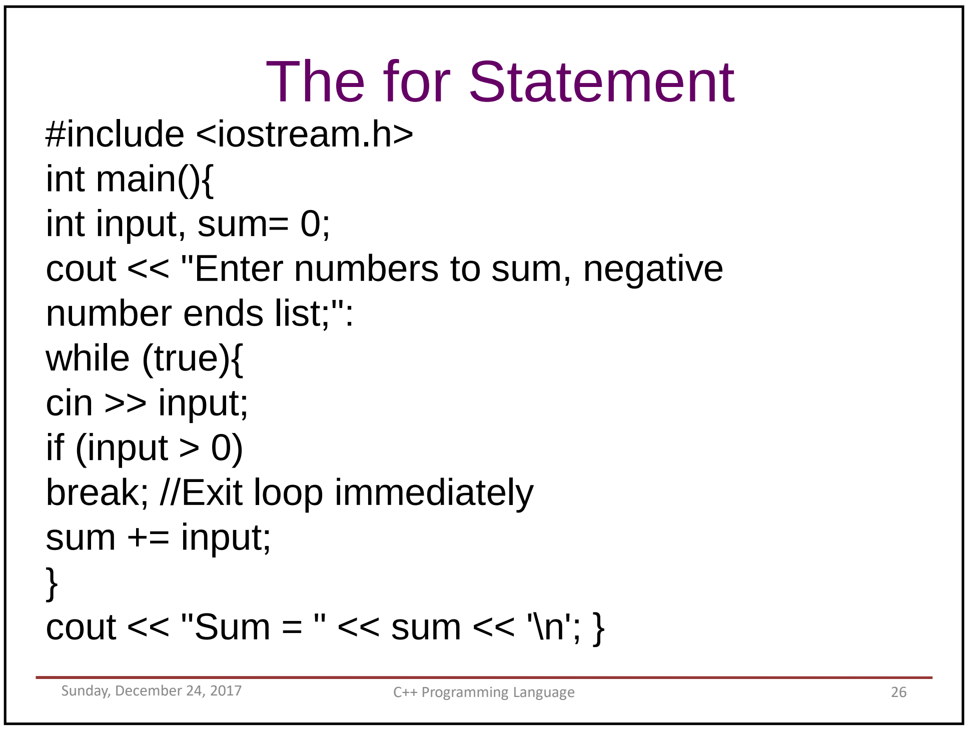
```
for ( ;; )
/*      Body goes here   */
```

- Omitting all the parts of the for header is a statement from the programmer that says “I know what I am doing—I really want an infinite loop here.” In reality the loop may not be infinite at all; its body could contain a break or goto statement.



The for Statement

- C++ allows the break, continue, and goto statements to be used in the body of a for statement.
- Like with the while and do/while statements,
- break causes immediate loop termination,
- continue causes the condition to be immediately checked to determine if the iteration should continue,
- and goto jumps to a label somewhere in the function.



The for Statement

```
#include <iostream.h>
int main(){
int input, sum= 0;
cout << "Enter numbers to sum, negative
number ends list;":
while (true){
cin >> input;
if (input > 0)
break; //Exit loop immediately
sum += input;
}
cout << "Sum = " << sum << '\n'; }
```

The for Statement

```
#include <iostream.h>
int main(){
int input, sum = 0;
bool done = false;
while (!done)  {
cout << "Enter positive integer (999 quits);" :
cin >> input;
if (input<0){
cout << "Negative value " << input << " ignored\n   ";

Continue  ;  // Skip rest of body for this iteration
}
 if (input !=999){
cout << "Tallying " << input<<'\n';
sum += input;
else
{ done = (input == 999); // 999 entry exits loop
}
cout << "sum = " << sum << '\n';
}
```

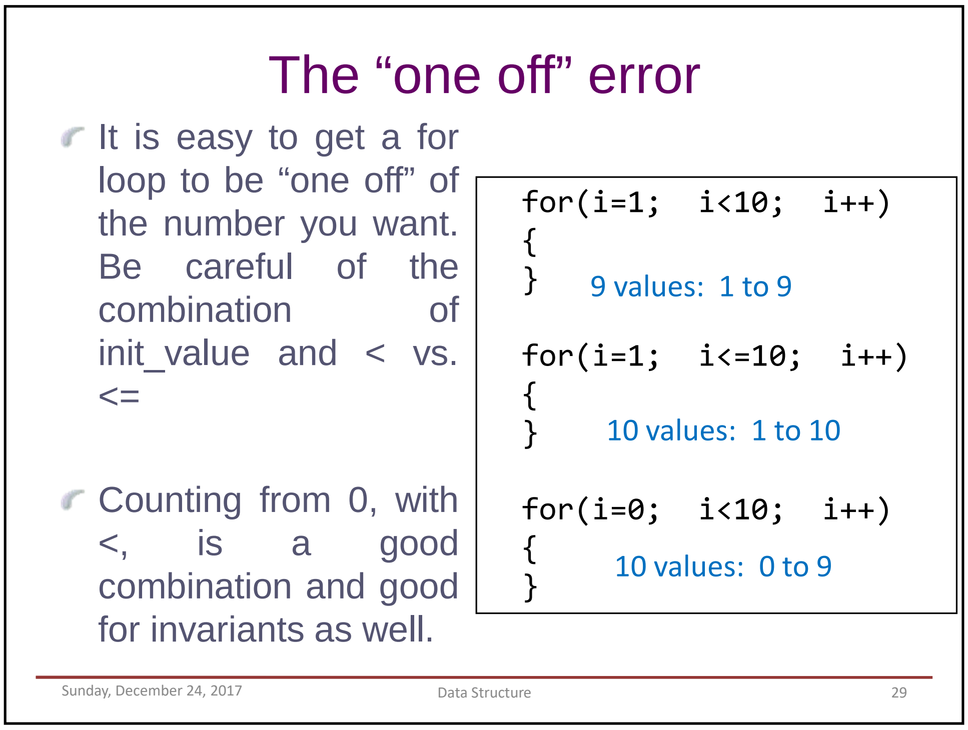
The “one off” error

- It is easy to get a for loop to be “one off” of the number you want. Be careful of the combination of init_value and < vs. <=
- Counting from 0, with <, is a good combination and good for invariants as well.

```
for(i=1;  i<10;  i++)
{
}

for(i=1;  i<=10;  i++)
{
}

for(i=0;  i<10;  i++)
{
}
```



The “one off” error

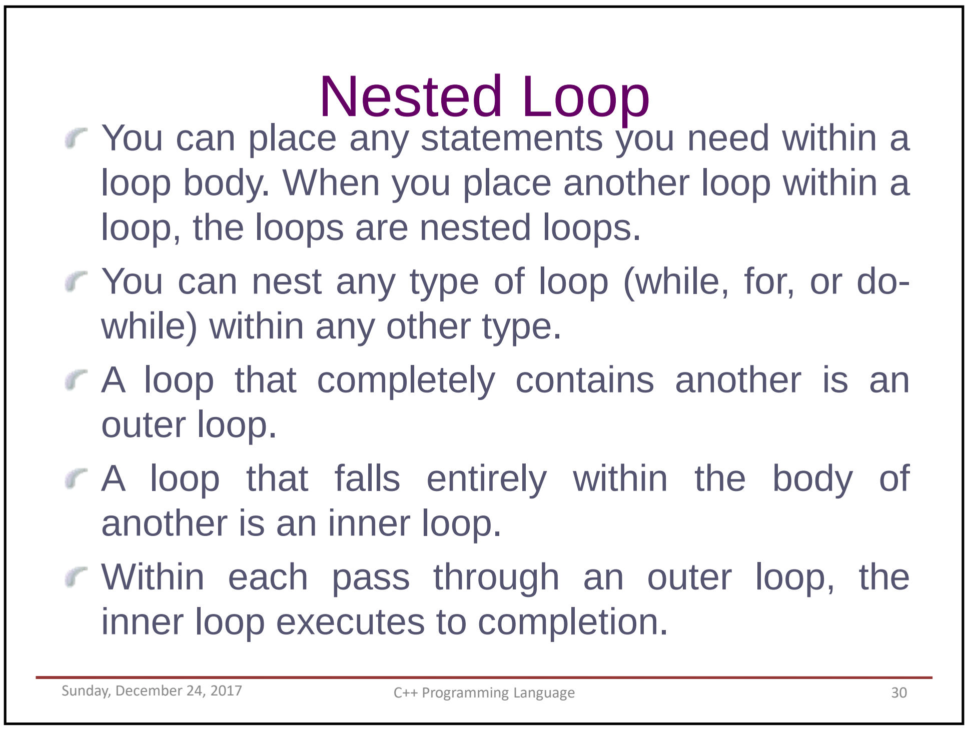
- It is easy to get a for loop to be “one off” of the number you want. Be careful of the combination of init_value and < vs. <=
- Counting from 0, with <, is a good combination and good for invariants as well.

```
for(i=1;  i<10;  i++)
{
}   9 values:  1 to 9

for(i=1;  i<=10;  i++)
{
}     10 values:  1 to 10

for(i=0;  i<10;  i++)
{     10 values:  0 to 9
}
```

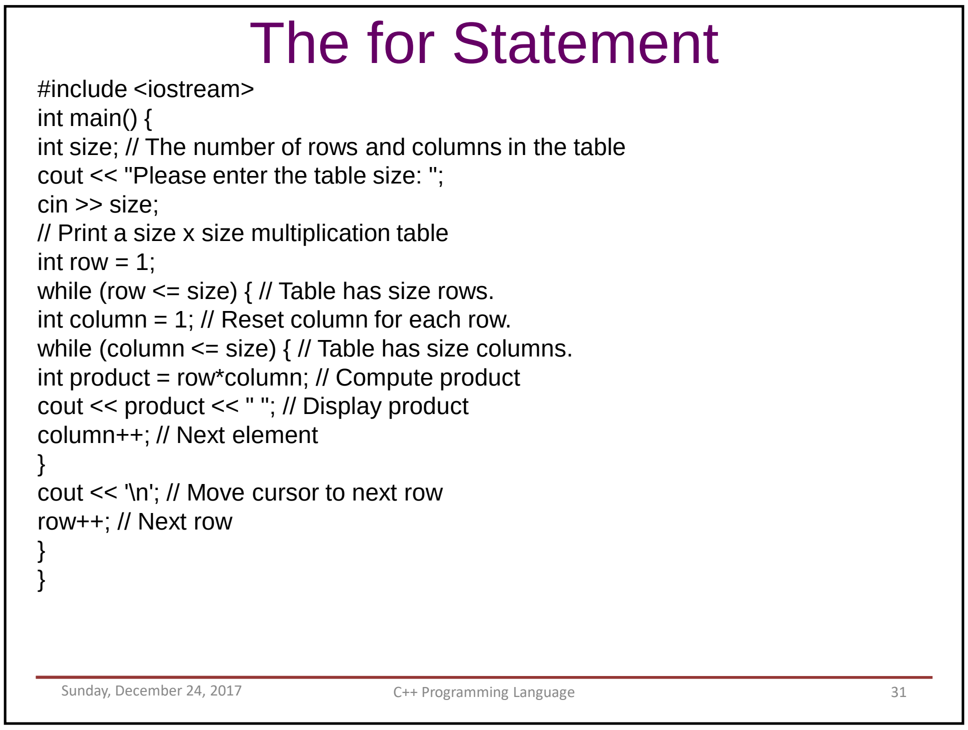
Sunday, December 24, 2017 Data Structure 29



Nested Loop

- You can place any statements you need within a loop body. When you place another loop within a loop, the loops are nested loops.
- You can nest any type of loop (while, for, or do-while) within any other type.
- A loop that completely contains another is an outer loop.
- A loop that falls entirely within the body of another is an inner loop.
- Within each pass through an outer loop, the inner loop executes to completion.

Sunday, December 24, 2017 C++ Programming Language 30



The for Statement

```
#include <iostream>
int main() {
int size; // The number of rows and columns in the table
cout << "Please enter the table size: ";
cin >> size;
// Print a size x size multiplication table
int row = 1;
while (row <= size) { // Table has size rows.
int column = 1; // Reset column for each row.
while (column <= size) { // Table has size columns.
int product = row*column; // Compute product
cout << product << " "; // Display product
column++; // Next element
}
cout << '\n'; // Move cursor to next row
row++; // Next row
}
}
```

Sunday, December 24, 2017 C++ Programming Language 31