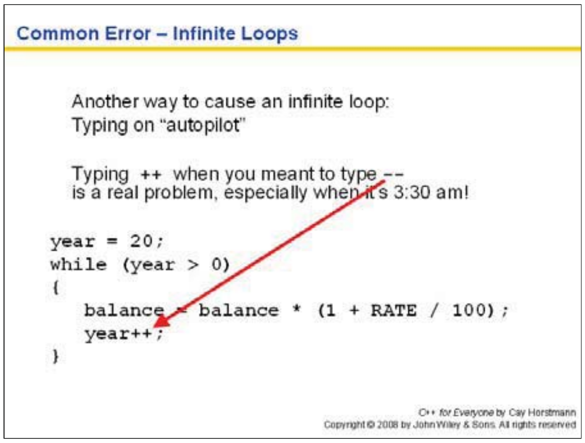


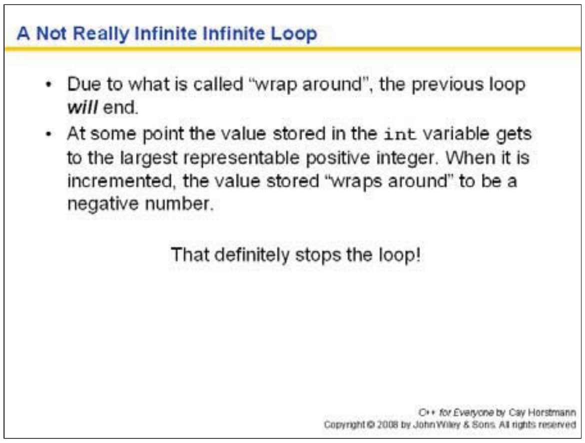
Common Error – Infinite Loops

Another way to cause an infinite loop:
Typing on "autopilot"

Typing `++` when you meant to type `--`
is a real problem, especially when it's 3:30 am!

```
year = 20;
while (year > 0)
{
   balance = balance * (1 + RATE / 100);
   year++;
}
```

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

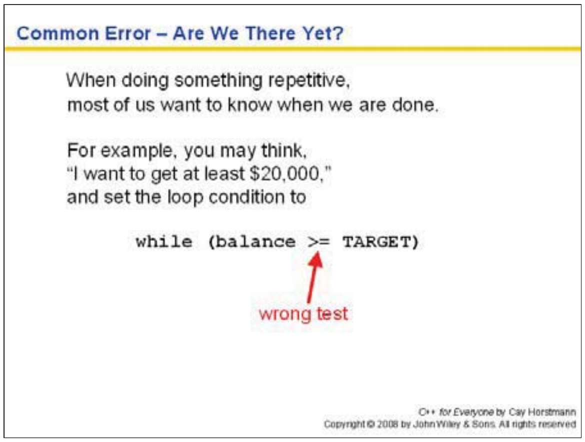


A Not Really Infinite Loop

- Due to what is called "wrap around", the previous loop ***will*** end.
- At some point the value stored in the `int` variable gets to the largest representable positive integer. When it is incremented, the value stored "wraps around" to be a negative number.

That definitely stops the loop!

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



Common Error – Are We There Yet?

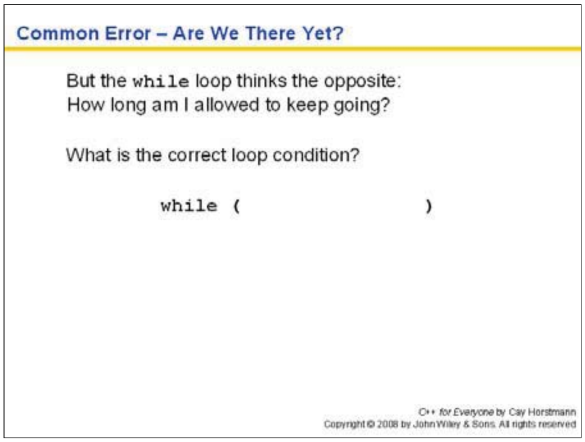
When doing something repetitive,
most of us want to know when we are done.

For example, you may think,
"I want to get at least $20,000,"
and set the loop condition to

```
while (balance >= TARGET)
```

wrong test

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



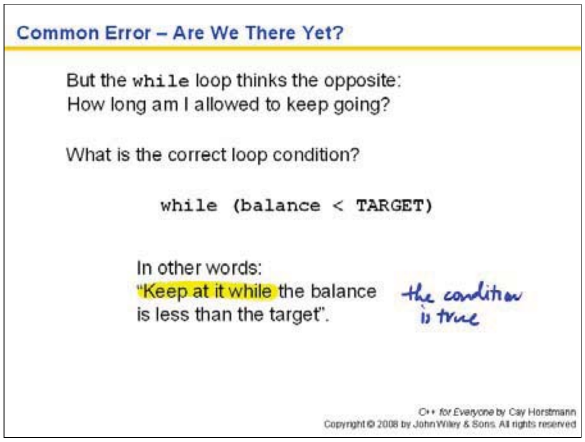
Common Error – Are We There Yet?

But the `while` loop thinks the opposite:
How long am I allowed to keep going?

What is the correct loop condition?

```
while (                 )
```

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



Common Error – Are We There Yet?

But the `while` loop thinks the opposite:
How long am I allowed to keep going?

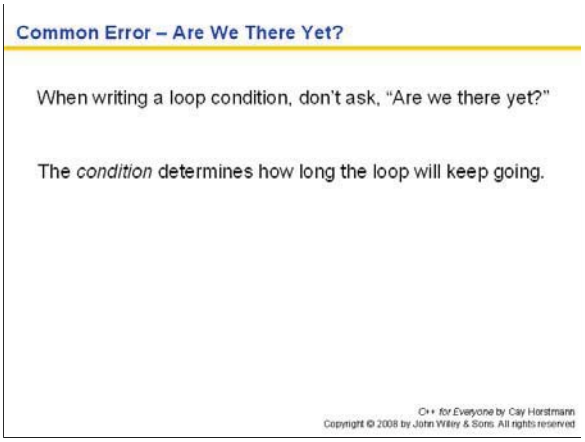
What is the correct loop condition?

```
while (balance < TARGET)
```

In other words:
"Keep at it while the balance
is less than the target".

the condition
is true

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

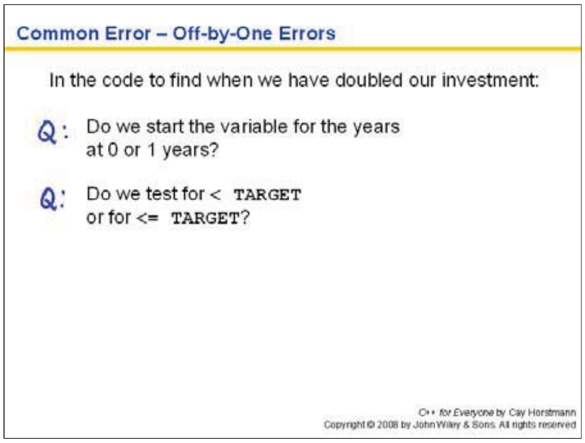


Common Error – Are We There Yet?

When writing a loop condition, don't ask, "Are we there yet?"

The *condition* determines how long the loop will keep going.

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



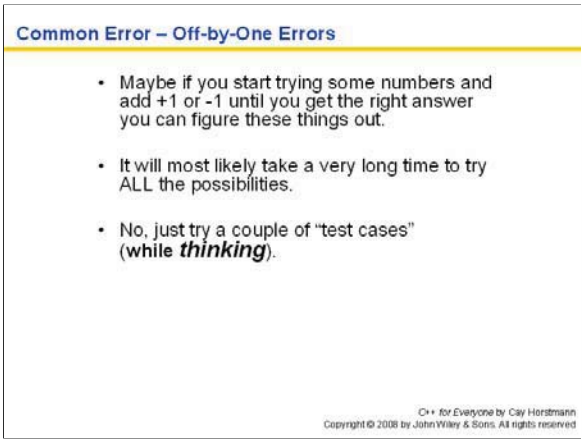
Common Error – Off-by-One Errors

In the code to find when we have doubled our investment:

Q: Do we start the variable for the years at 0 or 1 years?

Q: Do we test for < `TARGET` or for <= `TARGET`?

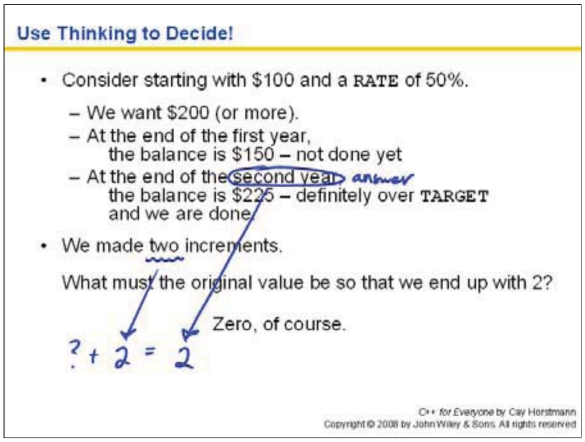
C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



Common Error – Off-by-One Errors

- Maybe if you start trying some numbers and add +1 or -1 until you get the right answer you can figure these things out.
- It will most likely take a very long time to try ALL the possibilities.
- No, just try a couple of "test cases" (**while *thinking***).

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



Use Thinking to Decide!

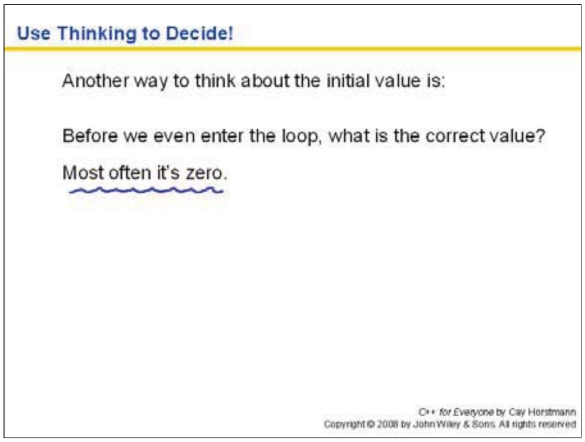
- Consider starting with $100 and a `RATE` of 50%.
  - We want $200 (or more).
  - At the end of the first year, the balance is $150 – not done yet
  - At the end of the second year answer the balance is $225 – definitely over `TARGET` and we are done
- We made two increments.

  What must the original value be so that we end up with 2?

  Zero, of course.

$? + 2 = 2$

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



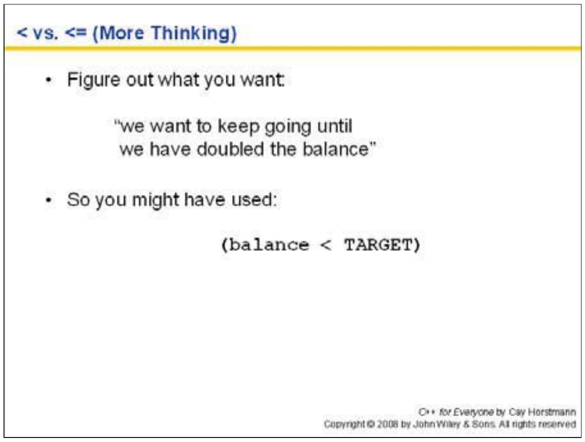
Use Thinking to Decide!

Another way to think about the initial value is:

Before we even enter the loop, what is the correct value?

Most often it's zero.

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



< vs. <= (More Thinking)

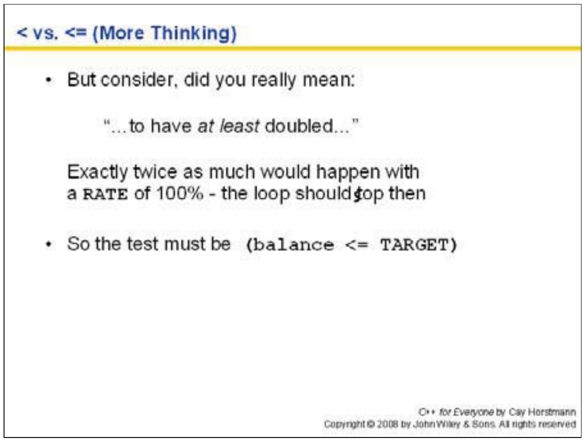
- Figure out what you want:

  "we want to keep going until we have doubled the balance"

- So you might have used:

  `(balance < TARGET)`

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



< vs. <= (More Thinking)

- But consider, did you really mean:

  "…to have *at least* doubled…"

  Exactly twice as much would happen with a `RATE` of 100% - the loop should stop then

- So the test must be `(balance <= TARGET)`

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

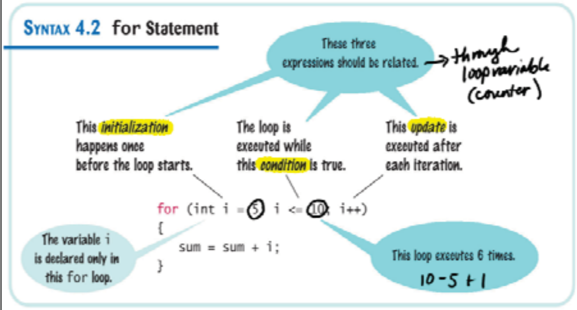
The `for` Loop (4.2)

You *"simply"* take 4,522 steps!!!

To execute statements a certain number of times

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

The `for` Loop



Syntax 4.2 for Statement

These three expressions should be related. → through loop variable (counter)

This *initialization* happens once before the loop starts.

The loop is executed while this *condition* is true.

This *update* is executed after each iteration.

```
for (int i = 5; i <= 10; i++)
{
   sum = sum + i;
}
```

The variable i is declared only in this for loop.

This loop executes 6 times.
10 - 5 + 1

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

The `for` Loop

```
for (initialization; check; update)
{
    statements
}
```

The *check* is some kind of test (the same as the condition in the `while` loop)

It is usually used to cause the *statements* to happen a certain number of times.

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

The `for` Loop

```
for (initialization; check; update)
{
    statements
}
```

The *statements* are repeatedly executed until the check is false.

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

The `for` Loop

```
for (initialization; check; update)
{
    statements
}
```

The *initialization* is code that happens once, before the check is made, in order to set up for counting how many times the *statements* will happen.

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

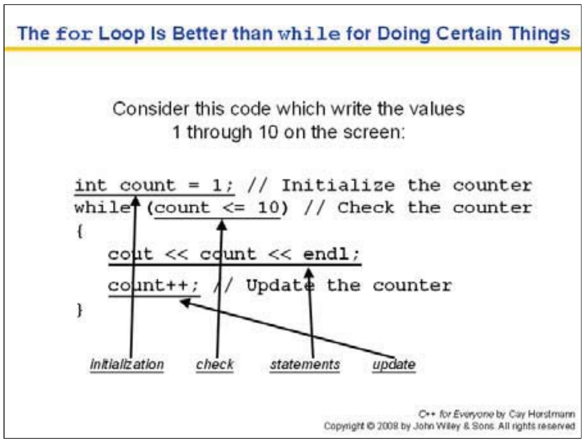
The `for` Loop

```
for (initialization; check; update)
{
    statements
}
```

The *update* is code that causes the check to eventually become false.

Usually it's incrementing or decrementing the loop variable.

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



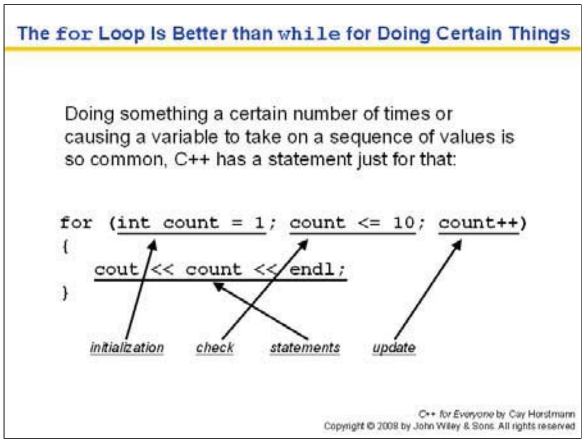
The `for` Loop Is Better than `while` for Doing Certain Things

Consider this code which write the values 1 through 10 on the screen:

```
int count = 1; // Initialize the counter
while (count <= 10) // Check the counter
{
   cout << count << endl;
   count++; // Update the counter
}
```

initialization
check
statements
update

C++ for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



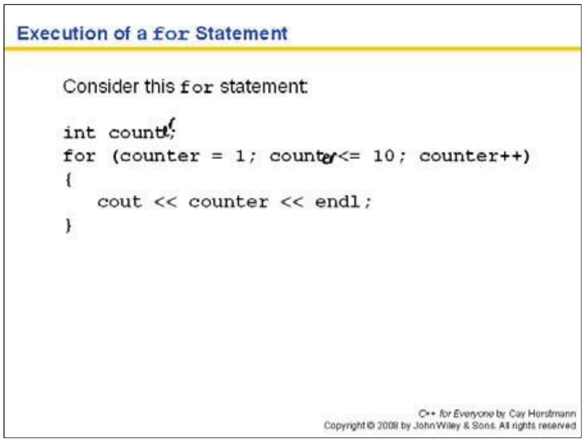
The `for` Loop Is Better than `while` for Doing Certain Things

Doing something a certain number of times or causing a variable to take on a sequence of values is so common, C++ has a statement just for that:

```
for (int count = 1; count <= 10; count++)
{
   cout << count << endl;
}
```

initialization
check
statements
update

C++ for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

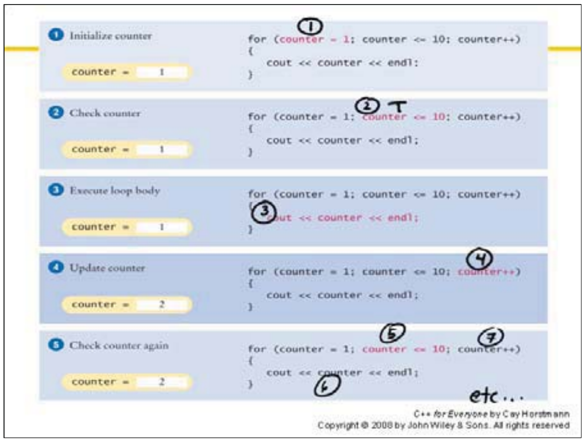


Execution of a `for` Statement

Consider this `for` statement:

```
int counter;
for (counter = 1; counter<= 10; counter++)
{
   cout << counter << endl;
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



1 Initialize counter

counter = 1

```
for (counter = 1; counter <= 10; counter++)
{
   cout << counter << endl;
}
```

2 Check counter

counter = 1

```
for (counter = 1; counter <= 10; counter++)
{
   cout << counter << endl;
}
```

3 Execute loop body

counter = 1

```
for (counter = 1; counter <= 10; counter++)
{
   cout << counter << endl;
}
```

4 Update counter

counter = 2

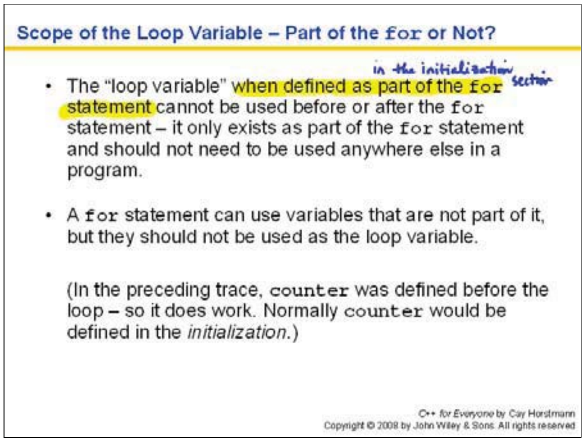
```
for (counter = 1; counter <= 10; counter++)
{
   cout << counter << endl;
}
```

5 Check counter again

counter = 2

```
for (counter = 1; counter <= 10; counter++)
{
   cout << counter << endl;
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

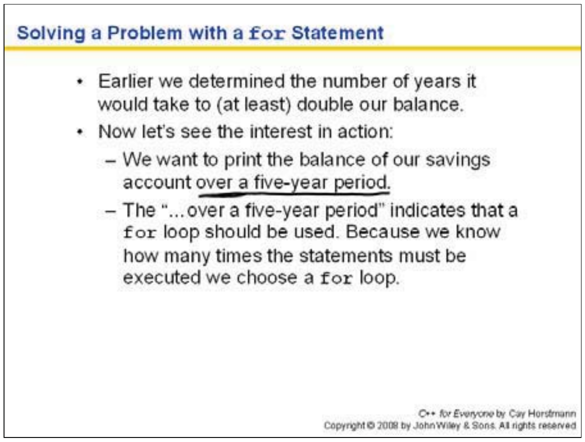


Scope of the Loop Variable – Part of the `for` or Not?

- The "loop variable" when defined as part of the `for` statement in the initialization section cannot be used before or after the `for` statement – it only exists as part of the `for` statement and should not need to be used anywhere else in a program.
- A `for` statement can use variables that are not part of it, but they should not be used as the loop variable.

(In the preceding trace, `counter` was defined before the loop – so it does work. Normally `counter` would be defined in the *initialization*.)

C++ for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



Solving a Problem with a `for` Statement

- Earlier we determined the number of years it would take to (at least) double our balance.
- Now let's see the interest in action:
  - We want to print the balance of our savings account over a five-year period.
  - The "...over a five-year period" indicates that a `for` loop should be used. Because we know how many times the statements must be executed we choose a `for` loop.

C++ for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

Solving a Problem with a `for` Statement

The output should look something like this:

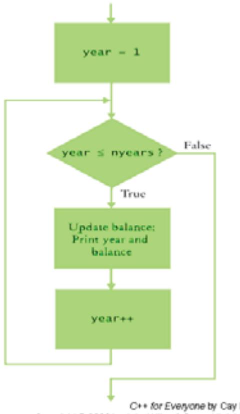
Year	Balance
1	10500.00
2	11025.00
3	11576.25
4	12155.06
5	12762.82

C++ for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

The `for` Loop

Flowchart of the investment calculation's `while` loop

Easily written using a `for` loop



C++ for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

Solving a Problem with a `for` Statement

Two statements should happen five times.
So use a `for` statement.

They are:

update balance
print year and balance

```
for (int year = 1; year <= nyears; year++)
{
    // update balance
    // print year and balance
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

The Modified Investment Program Using a `for` Loop

ch04/invtable.cpp

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
   const double RATE = 5;
   const double INITIAL_BALANCE = 10000;
   double balance = INITIAL_BALANCE;
   int nyears;

   cout << "Enter number of years: ";
   cin >> nyears;

   cout << fixed << setprecision(2);
   for (int year = 1; year <= nyears; year++)
   {
      balance = balance * (1 + RATE / 100);
      cout << setw(4) << year << setw(10) << balance << endl;
   }

   return 0;
}
```

C++ for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

The Modified Investment Program Using a `for` Loop

A run of the program:

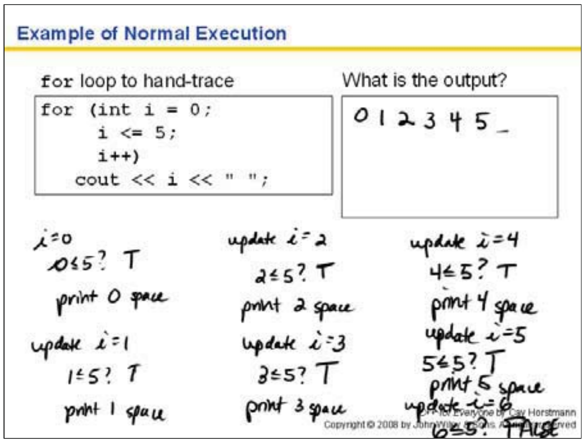
```
Enter number of years: 10
1 10500.00
2 11025.00
3 11576.25
4 12155.06
5 12762.82
6 13400.96
7 14071.00
8 14774.55
9 15513.28
10 16288.95
```

C++ for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

More `for` Examples

For each of the following, do a hand-trace.

C++ for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



Example of Normal Execution

`for` loop to hand-trace

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

What is the output?

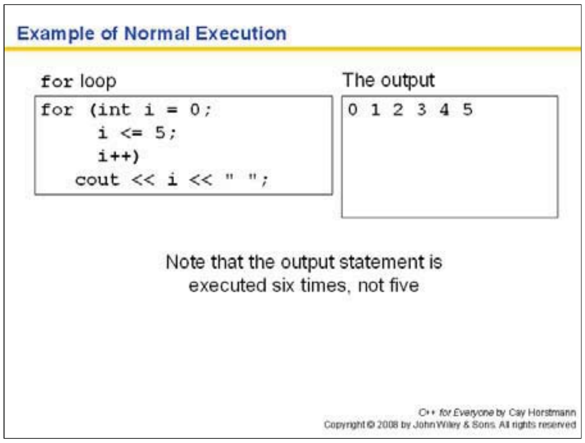
0 1 2 3 4 5 _

i=0
0≤5? T
print 0 space
update i=1
1≤5? T
print 1 space

update i=2
2≤5? T
print 2 space
update i=3
3≤5? T
print 3 space

update i=4
4≤5? T
print 4 space
update i=5
5≤5? T
print 5 space
update i=6
6≤5? FALSE

C++ for Everyone by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



Example of Normal Execution

`for` loop

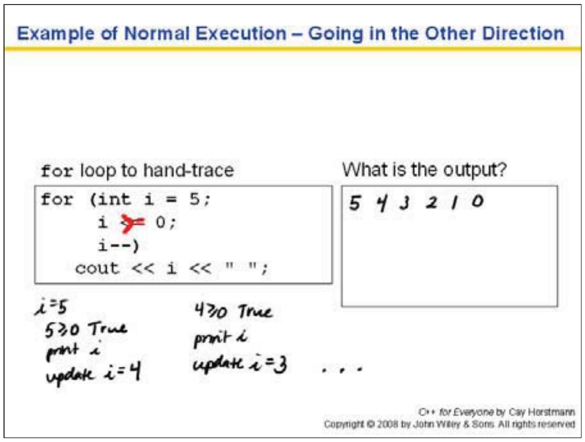
```
for (int i = 0;
      i <= 5;
      i++)
   cout << i << " ";
```

The output

```
0 1 2 3 4 5
```

Note that the output statement is executed six times, not five

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



Example of Normal Execution – Going in the Other Direction

`for` loop to hand-trace

```
for (int i = 5;
      i >= 0;
      i--)
   cout << i << " ";
```

What is the output?

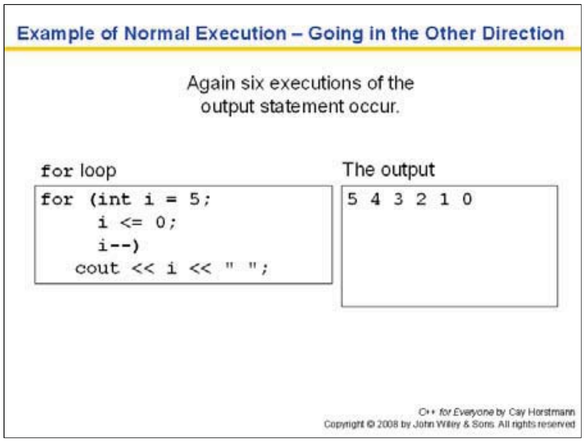
5 4 3 2 1 0

i=5
5≥0 True
print i
update i=4

4≥0 True
print i
update i=3

. . .

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



Example of Normal Execution – Going in the Other Direction

Again six executions of the output statement occur.

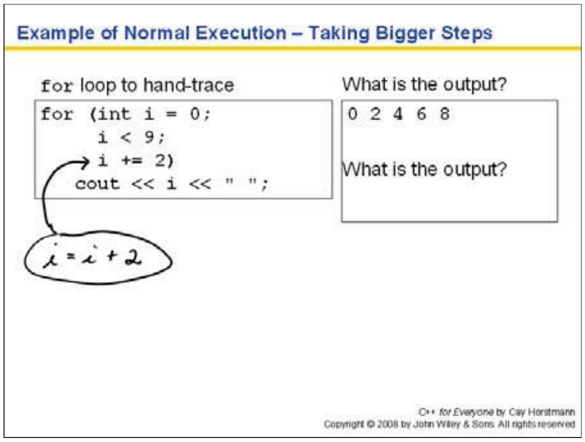
`for` loop

```
for (int i = 5;
      i <= 0;
      i--)
   cout << i << " ";
```

The output

```
5 4 3 2 1 0
```

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



Example of Normal Execution – Taking Bigger Steps

`for` loop to hand-trace

```
for (int i = 0;
      i < 9;
      i += 2)
   cout << i << " ";
```

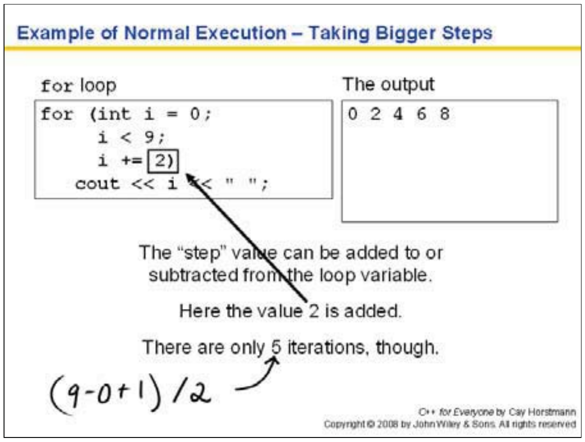
$i = i + 2$

What is the output?

```
0 2 4 6 8
```

What is the output?

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



Example of Normal Execution – Taking Bigger Steps

`for` loop

```
for (int i = 0;
      i < 9;
      i += 2)
   cout << i << " ";
```

The output

```
0 2 4 6 8
```

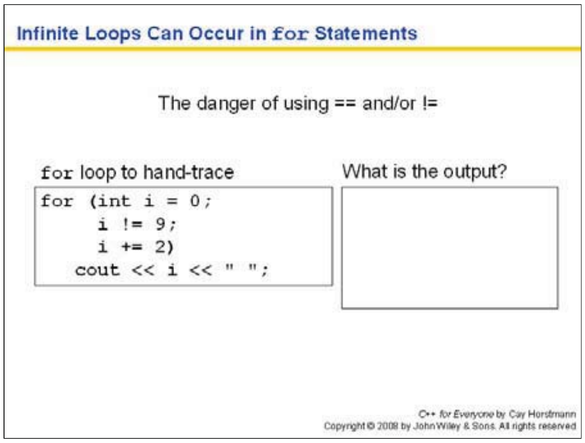
The "step" value can be added to or subtracted from the loop variable.

Here the value 2 is added.

There are only 5 iterations, though.

$(9-0+1)/2$

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



Infinite Loops Can Occur in for Statements

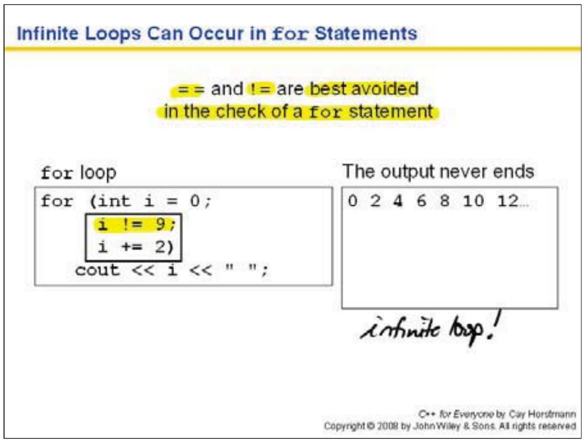
The danger of using == and/or !=

for loop to hand-trace

```
for (int i = 0;
     i != 9;
     i += 2)
  cout << i << " ";
```

What is the output?

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



Infinite Loops Can Occur in for Statements

== and != are best avoided
in the check of a for statement

for loop

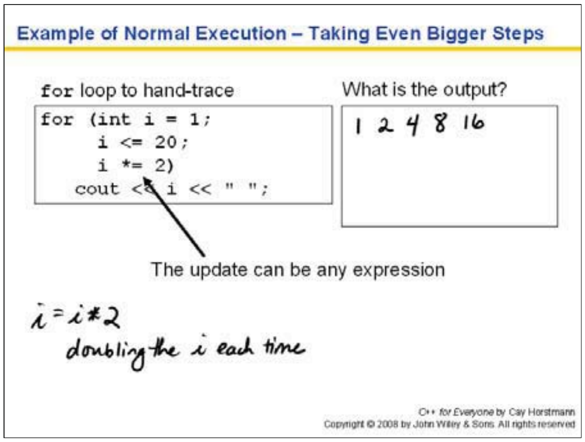
```
for (int i = 0;
     i != 9;
     i += 2)
  cout << i << " ";
```

The output never ends

```
0 2 4 6 8 10 12…
```

infinite loop!

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



Example of Normal Execution – Taking Even Bigger Steps

for loop to hand-trace

```
for (int i = 1;
     i <= 20;
     i *= 2)
  cout << i << " ";
```

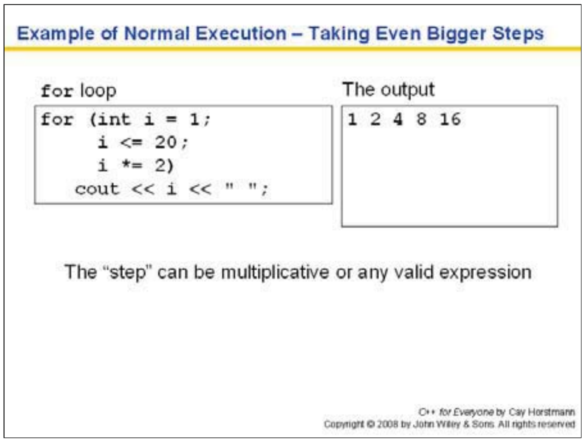
What is the output?

1 2 4 8 16

The update can be any expression

i=i*2
doubling the i each time

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



Example of Normal Execution – Taking Even Bigger Steps

for loop

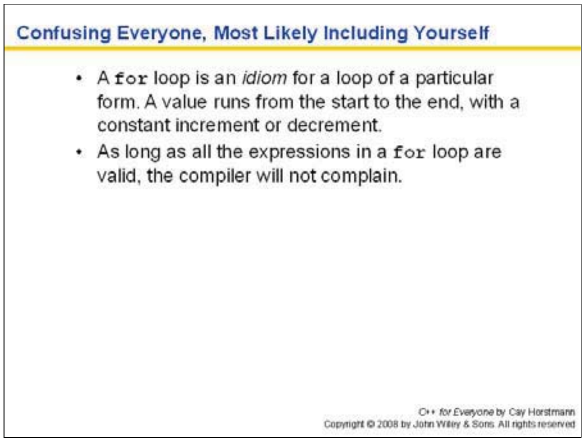
```
for (int i = 1;
     i <= 20;
     i *= 2)
  cout << i << " ";
```

The output

```
1 2 4 8 16
```

The “step” can be multiplicative or any valid expression

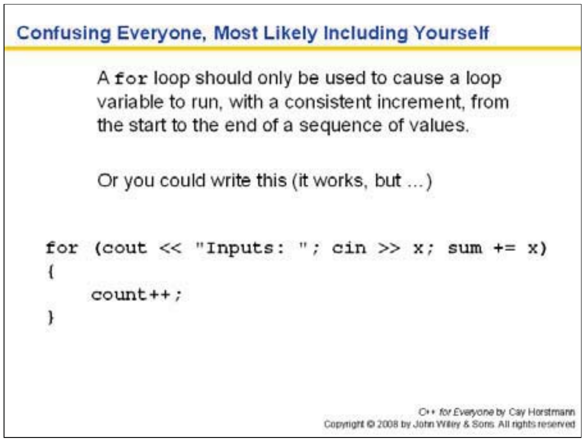
C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



Confusing Everyone, Most Likely Including Yourself

- A for loop is an *idiom* for a loop of a particular form. A value runs from the start to the end, with a constant increment or decrement.
- As long as all the expressions in a for loop are valid, the compiler will not complain.

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



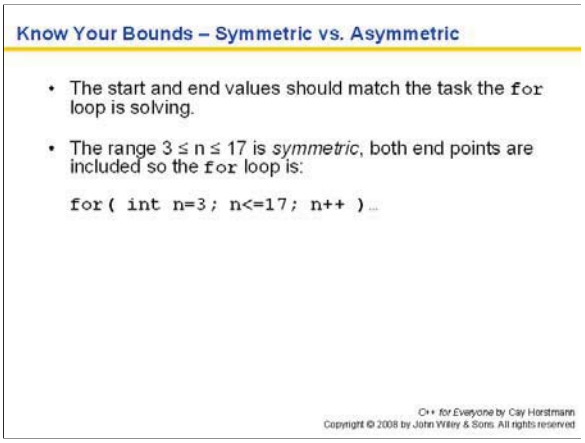
Confusing Everyone, Most Likely Including Yourself

A for loop should only be used to cause a loop variable to run, with a consistent increment, from the start to the end of a sequence of values.

Or you could write this (it works, but …)

```
for (cout << "Inputs: "; cin >> x; sum += x)
{
   count++;
}
```

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

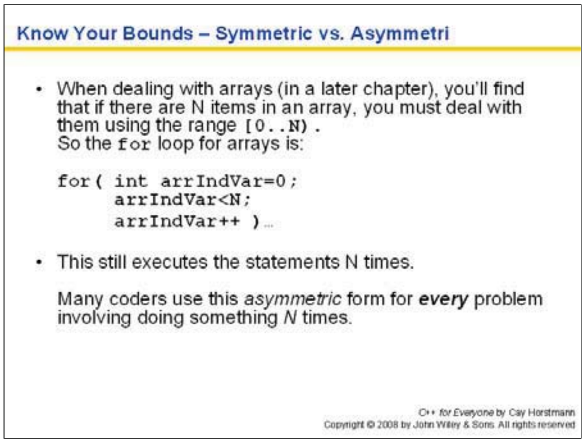


Know Your Bounds – Symmetric vs. Asymmetric

- The start and end values should match the task the `for` loop is solving.
- The range 3 ≤ n ≤ 17 is *symmetric*, both end points are included so the `for` loop is:

```
for( int n=3; n<=17; n++ )…
```

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



Know Your Bounds – Symmetric vs. Asymmetri

- When dealing with arrays (in a later chapter), you'll find that if there are N items in an array, you must deal with them using the range `[0..N)`.
  So the `for` loop for arrays is:

```
for( int arrIndVar=0;
     arrIndVar<N;
     arrIndVar++ )…
```

- This still executes the statements N times.

  Many coders use this *asymmetric* form for **every** problem involving doing something *N* times.

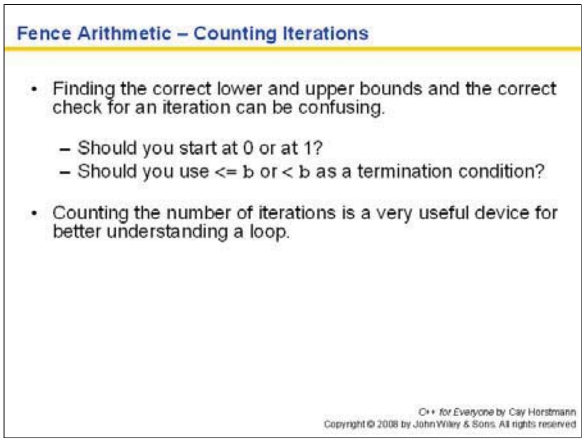
C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved

How Many Times Was That?

Fence arithmetic

Don't forget to count the first (or last)
"post number" that a loop variable takes on

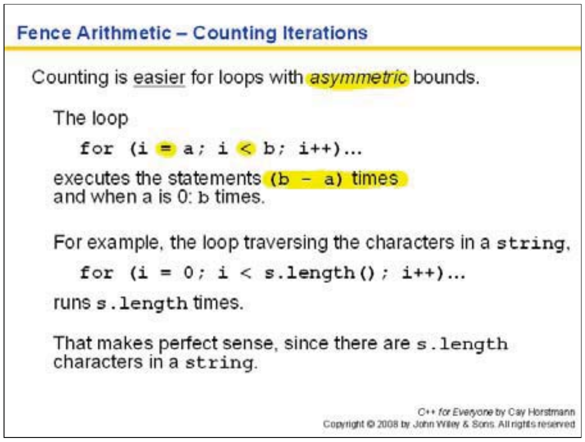
C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



Fence Arithmetic – Counting Iterations

- Finding the correct lower and upper bounds and the correct check for an iteration can be confusing.
  - Should you start at 0 or at 1?
  - Should you use <= b or < b as a termination condition?
- Counting the number of iterations is a very useful device for better understanding a loop.

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



Fence Arithmetic – Counting Iterations

Counting is easier for loops with *asymmetric* bounds.

The loop

```
for (i = a; i < b; i++)…
```

executes the statements `(b - a) times`
and when a is 0: `b` times.

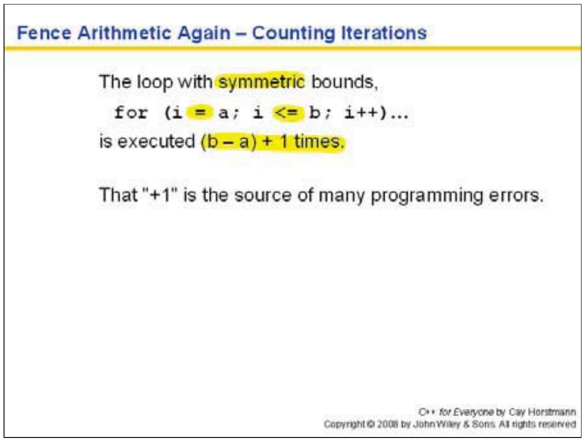
For example, the loop traversing the characters in a `string`,

```
for (i = 0; i < s.length(); i++)…
```

runs `s.length` times.

That makes perfect sense, since there are `s.length` characters in a `string`.

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved



Fence Arithmetic Again – Counting Iterations

The loop with symmetric bounds,

```
for (i = a; i <= b; i++)…
```

is executed (b – a) + 1 times.

That "+1" is the source of many programming errors.

C++ *for Everyone* by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved