# Chapter Four: Loops

## Chapter Goals

- To learn about the three types of loops:
  - `while`
  - `for`
  - `do`
- To avoid infinite loops and off-by-one errors
- To understand nested loops
- To implement programs that read and process data sets
- To use a computer for simulations

## What Is the Purpose of a Loop?

A loop is a statement that is used to:

execute one or more statements repeatedly until a goal is reached.

Sometimes these one-or-more statements will not be executed at all
—if that's the way to reach the goal

## The Three Loops in C++

C++ has these three looping statements:

```
while
for
do
```

## The `while` Loop (4.1)

Execute statements until a condition is true

## The `while` Loop

```
while (condition)
{
   statements
}
```

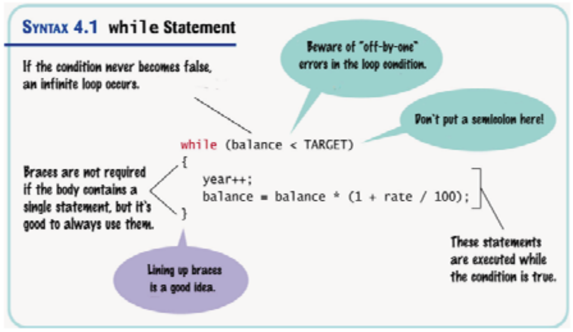
The *condition* is some kind of test
(the same as it was in the `if` statement in Chap. 3)

## The `while` Loop

```
while (condition)
{
    statements
}
```

The *statements* are repeatedly executed until the condition is `false`

## The `while` Loop


SYNTAX 4.1 while Statement
If the condition never becomes false, an infinite loop occurs.
Beware of "off-by-one" errors in the loop condition.
Don't put a semicolon here!
Braces are not required if the body contains a single statement, but it's good to always use them.
Lining up braces is a good idea.
These statements are executed while the condition is true.

```
while (balance < TARGET)
{
   year++;
   balance = balance * (1 + rate / 100);
}
```

## Using a Loop to Solve the Investment Problem.

The algorithm for an investment problem:

1. Start with a year value of 0 and a balance of $10,000.
2. Repeat the following steps
   **while the balance is less than $20,000:**  $A = P(1+rt)$
   $= P + Prt$
   - Add 1 to the year value.
   - Multiply the balance value by 1.05 (a 5 percent increase). $= P + I$
3. Report the final year value as the answer.

"Repeat .. while" in the problem indicates a loop is needed. To reach the goal of being able to report the final year value, adding and multiplying must be repeated some unknown number of times.

## Using a Loop to Solve the Investment Problem.

The statements to be controlled are:
- Incrementing the `year` variable
- Updating the `balance` variable using a `const` for the `RATE`

```
year++;
balance = balance * (1.+ RATE / 100);
```

## Using a Loop to Solve the Investment Problem.

The condition, which indicates when to ***stop*** executing the statements, is this test:

```
(balance < TARGET)
```

Want to stop the loop when $balance \geq TARGET$

$balance \geq TARGET$ is true

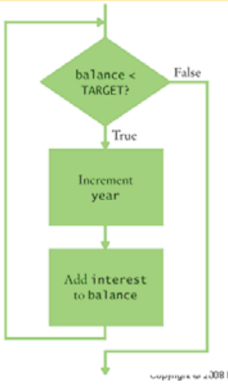
when $balance < TARGET$ is false

## Using a Loop to Solve the Investment Problem.

Here is the complete `while` statement:

```
while (balance < TARGET)
{
   year++;
   balance = balance * (1.+ RATE / 100);
}
```

## Flowchart of the Investment Calculation's `while` Loop

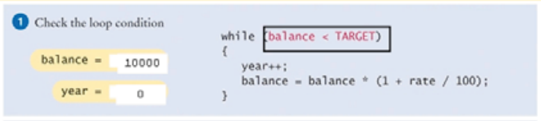


## The Complete Investment Program

ch04/doublinv.cpp

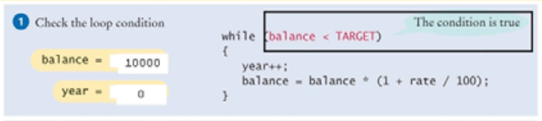
```
#include <iostream>
using namespace std;

int main()
{
   const double RATE = 5;
   const double INITIAL_BALANCE = 10000;
   const double TARGET = 2 * INITIAL_BALANCE;

   double balance = INITIAL_BALANCE;
   int year = 0;

   while (balance < TARGET)
   {
      year++;
      balance = balance * (1 + RATE / 100);
   }

   cout << "The investment doubled after "
        << year << " years." << endl;

   return 0;
}
```

## Program Run



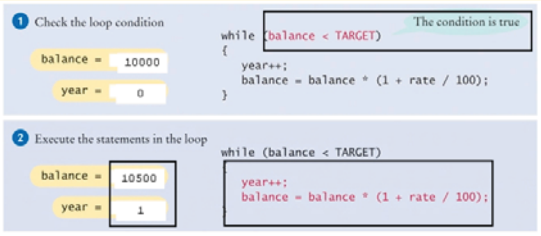
```
while (balance < TARGET)
{
   year++;
   balance = balance * (1 + rate / 100);
}
```

## Program Run



```
while (balance < TARGET)
{
   year++;
   balance = balance * (1 + rate / 100);
}
```

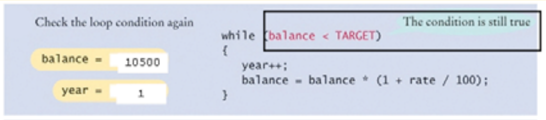
## Program Run



```
while (balance < TARGET)
{
   year++;
   balance = balance * (1 + rate / 100);
}
```

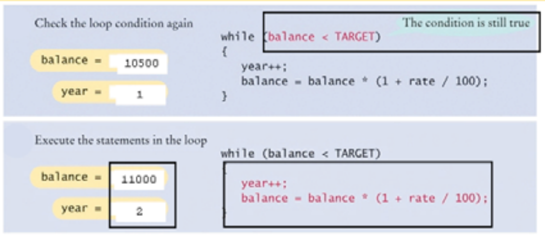
```
while (balance < TARGET)
   year++;
   balance = balance * (1 + rate / 100);
```

## Program Run



```
while (balance < TARGET)
{
   year++;
   balance = balance * (1 + rate / 100);
}
```

## Program Run

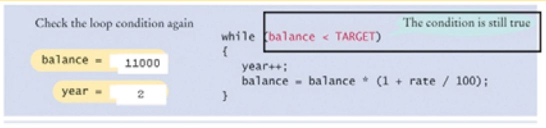

Check the loop condition again
balance = 10500
year = 1
The condition is still true

```
while (balance < TARGET)
{
   year++;
   balance = balance * (1 + rate / 100);
}
```

Execute the statements in the loop
balance = 11000
year = 2

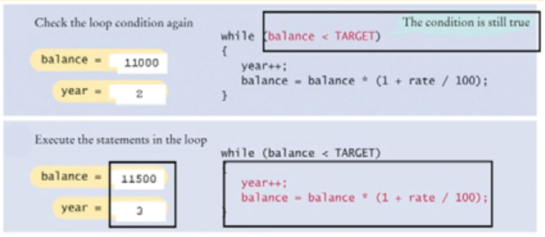
```
while (balance < TARGET)
   year++;
   balance = balance * (1 + rate / 100);
```

## Program Run



```
while (balance < TARGET)
{
   year++;
   balance = balance * (1 + rate / 100);
}
```

## Program Run


Check the loop condition again
balance = 11000
year = 2
The condition is still true

```
while (balance < TARGET)
{
   year++;
   balance = balance * (1 + rate / 100);
}
```

Execute the statements in the loop
balance = 11500
year = 3

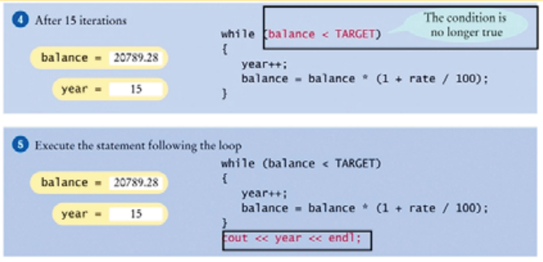
```
while (balance < TARGET)
   year++;
   balance = balance * (1 + rate / 100);
```

## Program Run

...This process continues
for 15 iterations...

a "time" through the loop

## Program Run

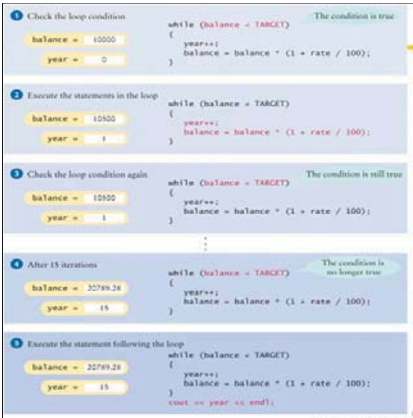


```
while (balance < TARGET)
{
   year++;
   balance = balance * (1 + rate / 100);
}
```

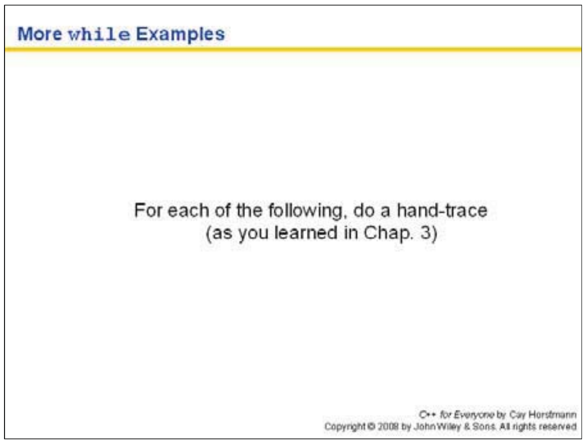
```
while (balance < TARGET)
{
   year++;
   balance = balance * (1 + rate / 100);
}
cout << year << endl;
```

The final output indicates
that the investment
doubled in 15 years.

## Program Run

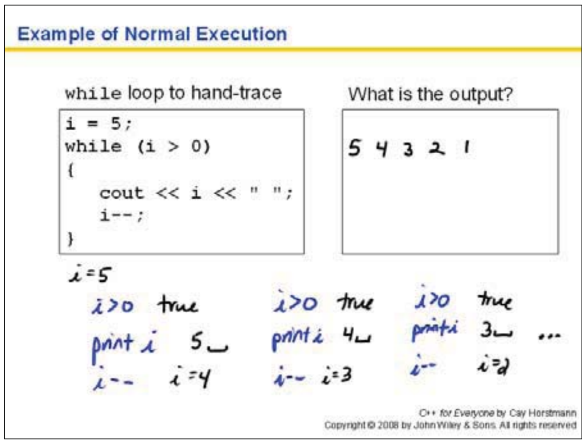

1 Check the loop condition
balance = 10000
year = 0
The condition is true
2 Execute the statements in the loop
balance = 10500
year = 1
3 Check the loop condition again
balance = 10500
year = 1
The condition is still true
4 After 15 iterations
balance = 20789.28
year = 15
The condition is no longer true
5 Execute the statement following the loop
balance = 20789.28
year = 15

```
while (balance < TARGET)
{
   year++;
   balance = balance * (1 + rate / 100);
}
cout << year << endl;
```



## More while Examples

For each of the following, do a hand-trace
(as you learned in Chap. 3)



## Example of Normal Execution

while loop to hand-trace

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

What is the output?

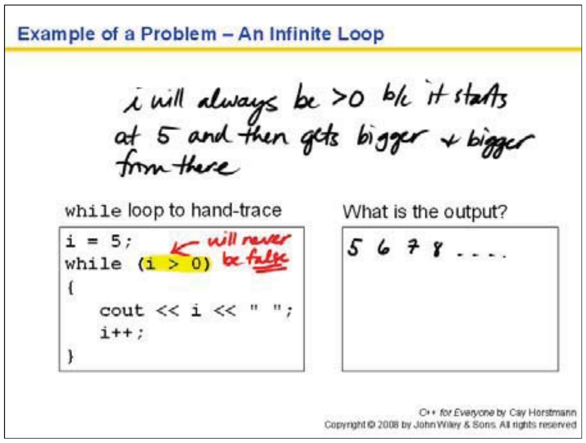
5 4 3 2 1

i=5

i>0 true
print i 5␣
i-- i=4

i>0 true
print i 4␣
i-- i=3

i>0 true
print i 3␣ ...
i-- i=2



## Example of a Problem – An Infinite Loop

i will always be >0 b/c it starts at 5 and then gets bigger + bigger from there

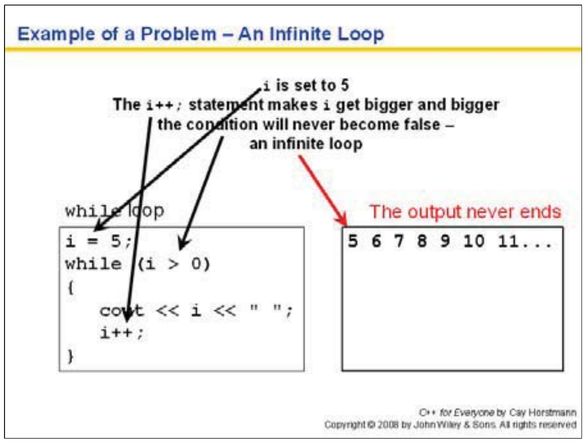
while loop to hand-trace

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

will never be false

What is the output?

5 6 7 8 . . . .



## Example of a Problem – An Infinite Loop

i is set to 5

The i++; statement makes i get bigger and bigger

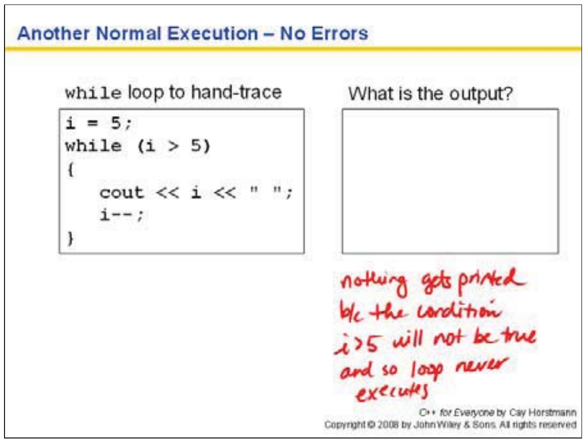
the condition will never become false –
an infinite loop

while loop

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

The output never ends

```
5 6 7 8 9 10 11...
```



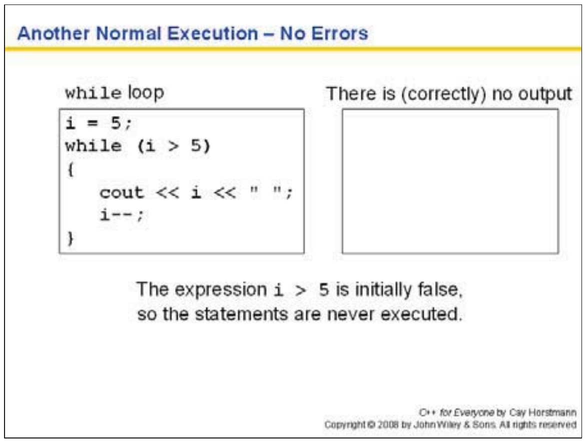
## Another Normal Execution – No Errors

while loop to hand-trace

```
i = 5;
while (i > 5)
{
   cout << i << " ";
   i--;
}
```

What is the output?

nothing gets printed b/c the condition i>5 will not be true and so loop never executes



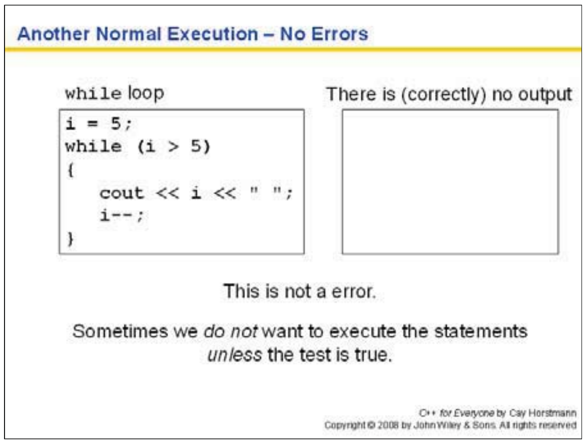
## Another Normal Execution – No Errors

while loop

```
i = 5;
while (i > 5)
{
   cout << i << " ";
   i--;
}
```

There is (correctly) no output

The expression i > 5 is initially false,
so the statements are never executed.



## Another Normal Execution – No Errors

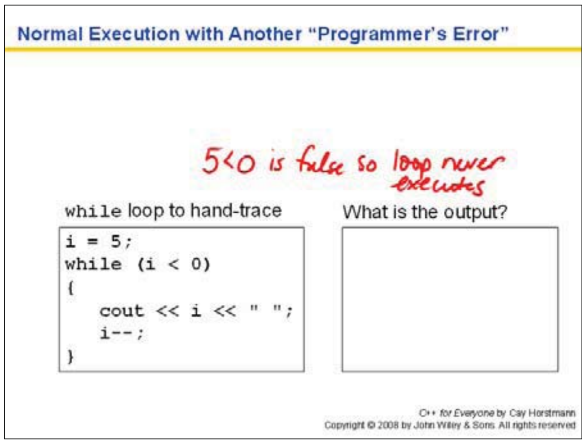
while loop

```
i = 5;
while (i > 5)
{
   cout << i << " ";
   i--;
}
```

There is (correctly) no output

This is not a error.

Sometimes we *do not* want to execute the statements *unless* the test is true.



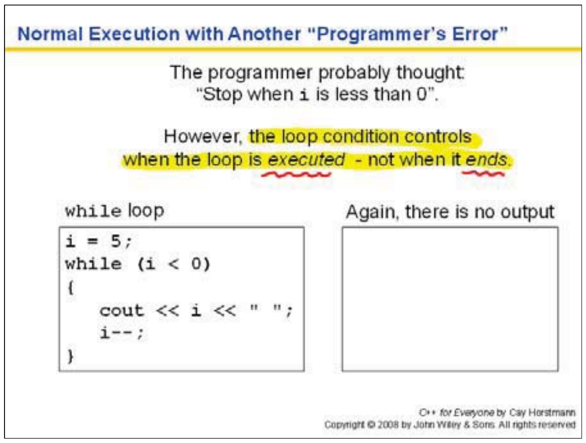
## Normal Execution with Another "Programmer's Error"

5<0 is false so loop never executes

while loop to hand-trace

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

What is the output?



## Normal Execution with Another "Programmer's Error"

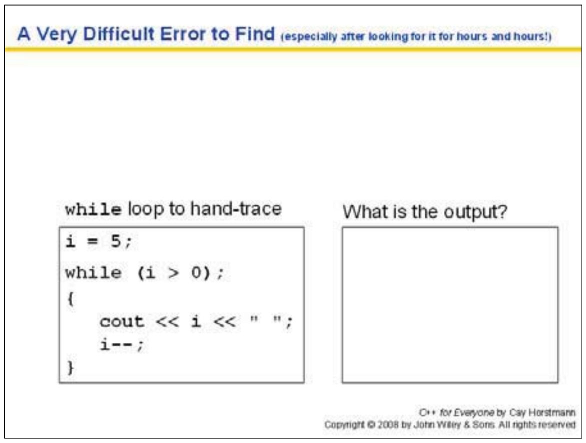
The programmer probably thought:
"Stop when `i` is less than 0".

However, the loop condition controls when the loop is *executed* - not when it *ends*.

while loop

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

Again, there is no output

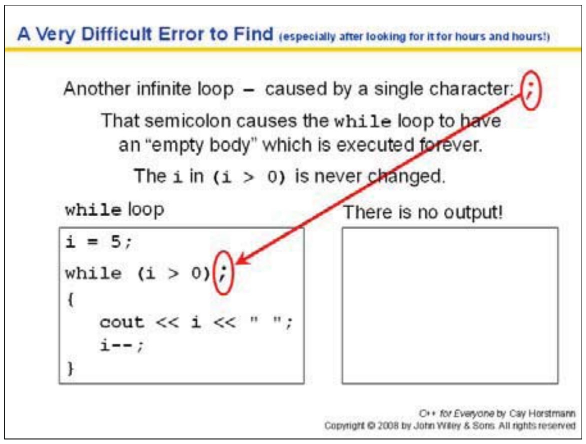


## A Very Difficult Error to Find (especially after looking for it for hours and hours!)

while loop to hand-trace

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

What is the output?



## A Very Difficult Error to Find (especially after looking for it for hours and hours!)

Another infinite loop – caused by a single character: ;

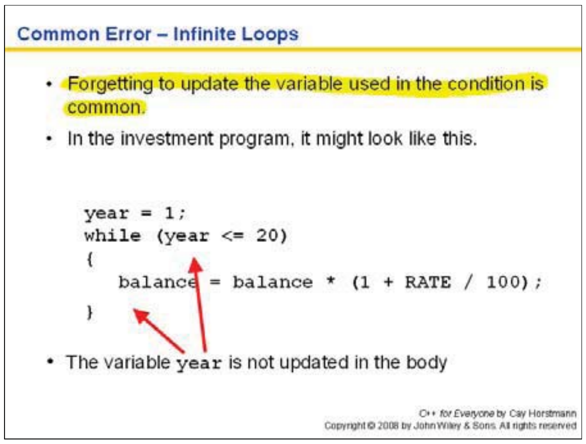
That semicolon causes the `while` loop to have an "empty body" which is executed forever.

The `i` in `(i > 0)` is never changed.

while loop

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

There is no output!

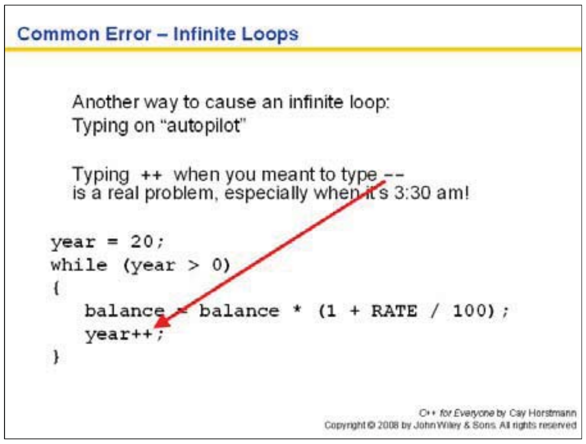


## Common Error – Infinite Loops

- Forgetting to update the variable used in the condition is common.
- In the investment program, it might look like this.

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

- The variable `year` is not updated in the body

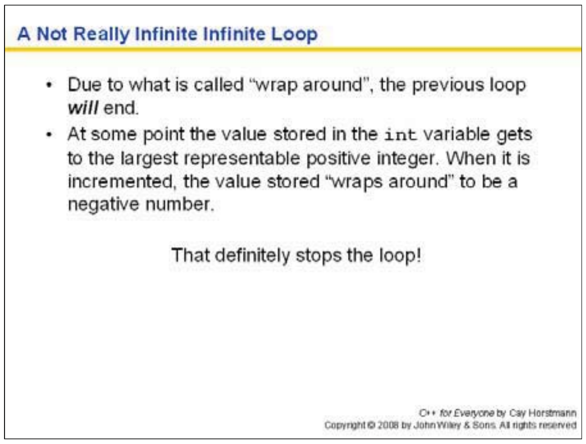


## Common Error – Infinite Loops

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!

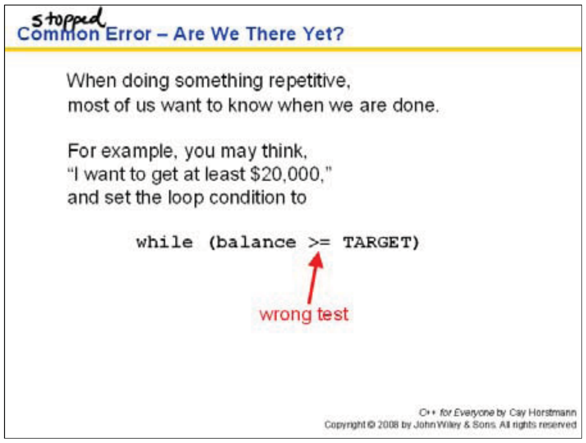
```
year = 20;
while (year > 0)
{
   balance = balance * (1 + RATE / 100);
   year++;
}
```



## 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!



stopped

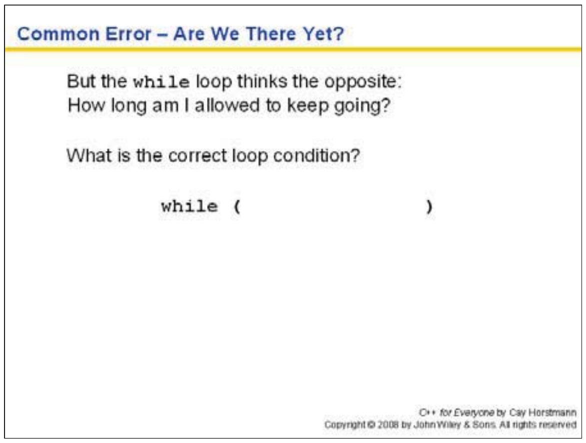
## Common Error – Are We There Yet?

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

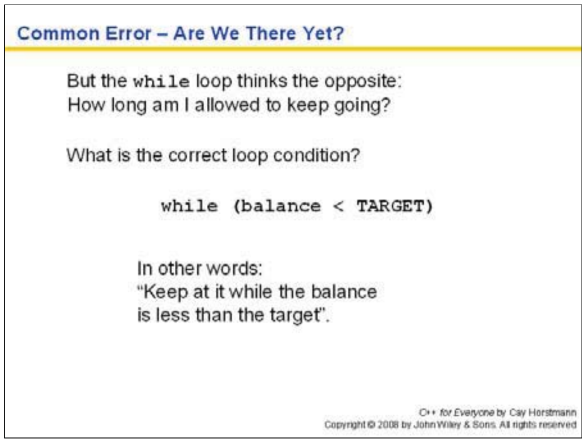


## Common Error – Are We There Yet?

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

What is the correct loop condition?

```
while (                    )
```



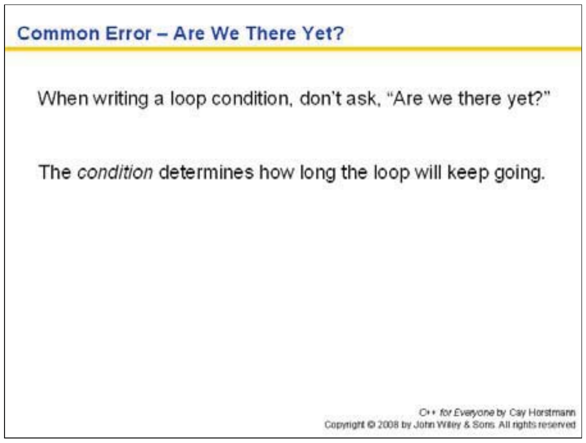
## Common Error – Are We There Yet?

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".



## 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.