© photo75/iStockphoto.

Chapter Four: Loops

Chapter Goals

- To implement `while`, `for` and `do…while` loops
- 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

Topic 1

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 statements will not be executed at all
—if that’s the way to reach the goal

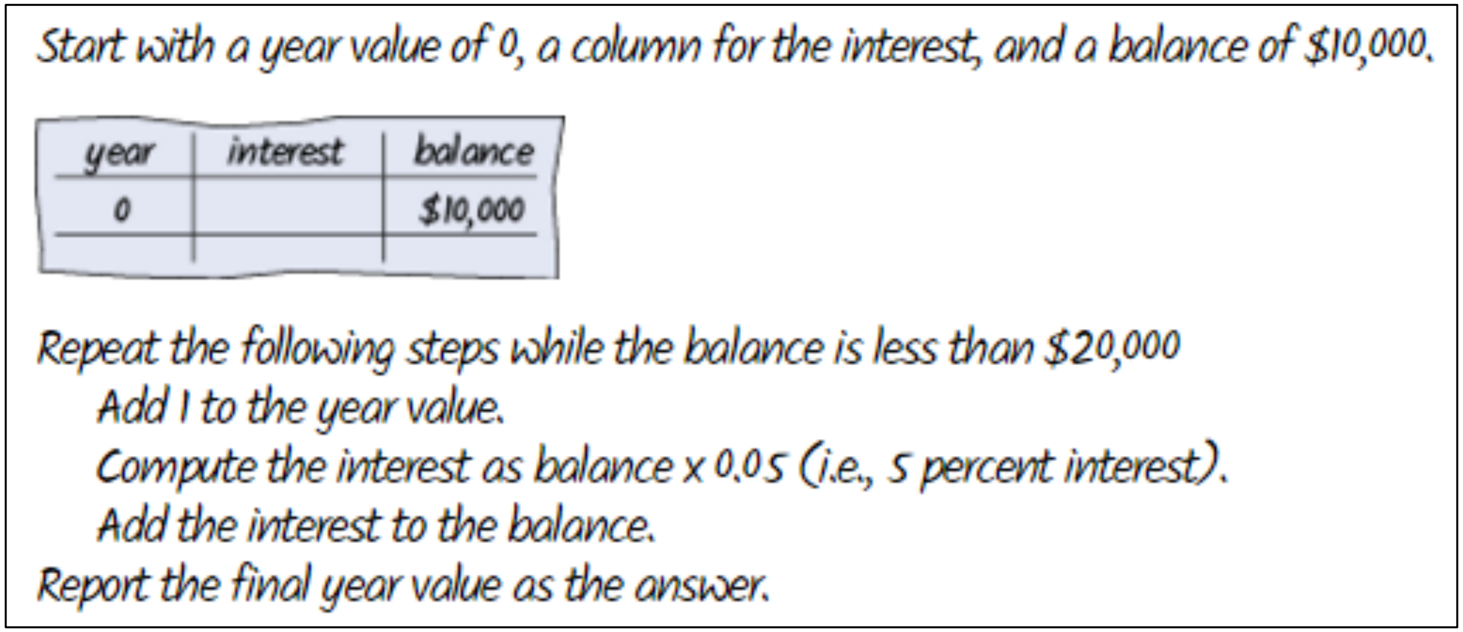
The Three Loops in C++

C++ has three looping statements:

```
while()
for()
do {} while()
```

The `while` Loop

- Chapter 1 had an example of an algorithm needing a loop
  - "repeat … while the balance is less…"



Start with a year value of 0, a column for the interest, and a balance of $10,000.

year	interest	balance
0		$10,000

```
Repeat the following steps while the balance is less than $20,000
    Add 1 to the year value.
    Compute the interest as balance x 0.05 (i.e., 5 percent interest).
    Add the interest to the balance.
Report the final year value as the answer.
```

The `while` Loop template

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

The *condition* is some kind of test (just like the **`if`** statement)

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

Using a Loop to Solve an Investment Problem

An investment problem:

Starting with $10,000, how many years until we have at least $20,000, at 5% interest?

The algorithm:

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**:
   - Add 1 to the year value.
   - Compute the interest by multiplying the balance value by 0.05 (5 percent interest) (will be a `const`, of course).
   - Add the interest to the balance.
3. Report the final year value as the answer.

The Complete Investment Program

sec01/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++;
      double interest = balance * RATE / 100;
      balance = balance + interest;
   }

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

   return 0;
}
```

Program Run

```
The investment doubled after 15 years.
```

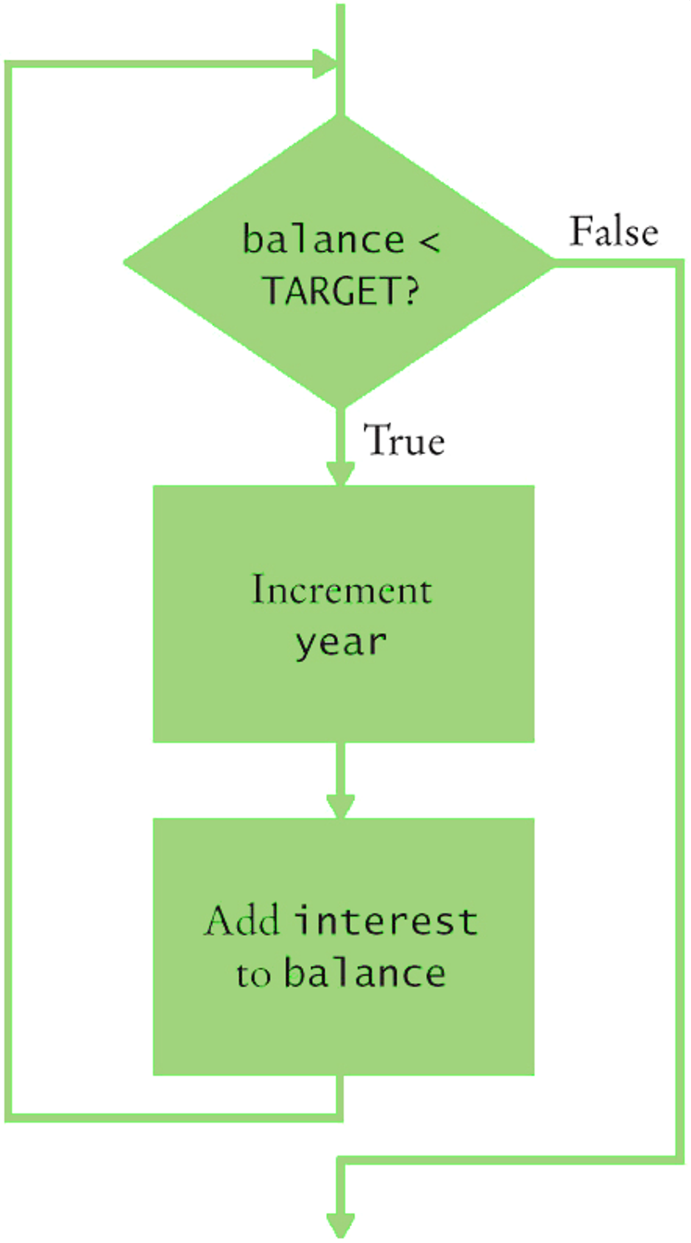
Program Run

…the values are updated for 15 iterations…

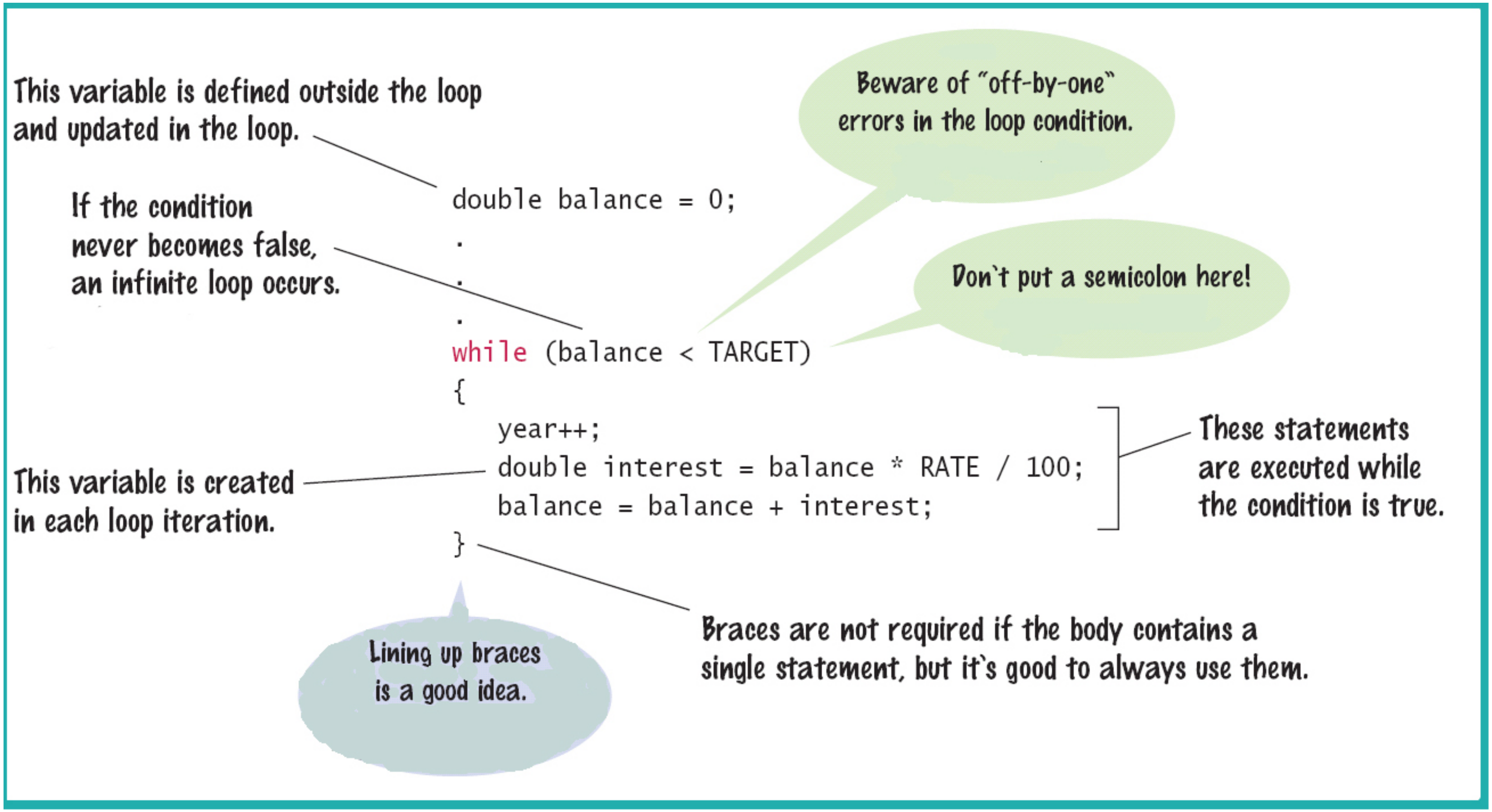
…until the **`balance`** is finally(!) over $20,000 and the **`while()`** test becomes **`false`**.

before entering while's body		at the end of while's body		
balance	year	interest	balance	year
10000.00	0	500.00	10500.00	1
10500.00	1	525.00	11025.00	2
11025.00	2	551.25	11576.25	3
11576.25	3	578.81	12155.06	4
12155.06	4	607.75	12762.82	5
12762.82	5	638.14	13400.96	6
13400.96	6	670.05	14071.00	7
14071.00	7	703.55	14774.55	8
14774.55	8	738.73	15513.28	9
15513.28	9	775.66	16288.95	10
16288.95	10	814.45	17103.39	11
17103.39	11	855.17	17958.56	12
17958.56	12	897.93	18856.49	13
18856.49	13	942.82	19799.32	14
19799.32	14	989.97	20789.28	15
20789.28	15	while statement is over		

Flowchart of the Investment Calculation `while` Loop



The `while` Statement



```
double balance = 0;
.
.
.
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

This variable is defined outside the loop and updated in the loop.
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!
These statements are executed while the condition is true.
This variable is created in each loop iteration.
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.

`while` Loop Examples: Table 1

Loop (all preceded by `i=5;`)	Output	Explanation

`while (i > 0)`
`{ cout << i << " "; i--;`
`}`
	5 4 3 2 1	When i is 0, the loop condition is false, and the loop ends.

`while (i > 0)`
`{ cout << i << " "; i++;`
`}`
	5 6 7 8 9 10 11 ...	The i++ statement is an error causing an "infinite loop" (see Common Error 4.1).

`while (i > 5)`
`{ cout << i << " "; i--;`
`}`
	(No output)	The statement i > 5 is false, and the loop is never executed.

`while (i < 0)`
`{ cout << i << " ";`
`  i--;`
`}`
	(No output)	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 (see Common Error 4.2).

`while (i > 0);`
`{ cout << i << " ";`
`i--;`
`}`
	(No output, program does not terminate)	Note the semicolon before the {. This loop has an empty body. It runs forever, checking whether i > 0 and doing nothing in the body.

Example of Normal Execution

`while` loop to hand-trace

What is the output?

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

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

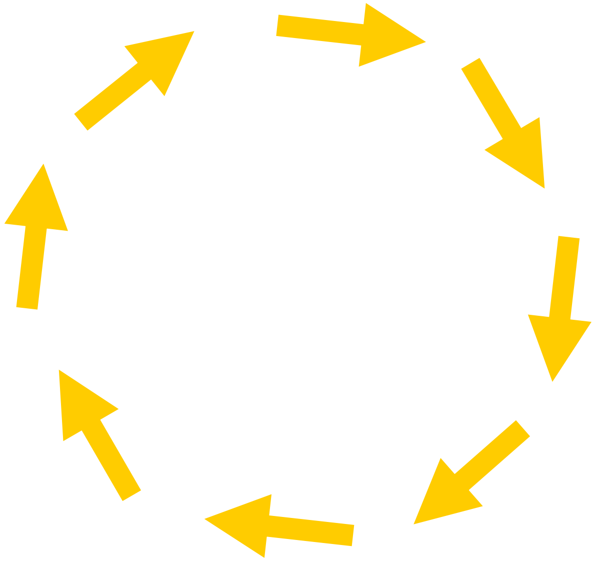
```
i = 5;
while (i > 0)
{
   cout << i << " ";
   i++;
}
```

The output never ends

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

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

Another Programmer Error

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

What is the output?

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

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

What is the output?

The Answer: Difficult Error to Find

Another infinite loop – caused by the semicolon after the parentheses.

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 – Off-by-One Errors

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

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

Do we test for < **`TARGET`**

or for <= **`TARGET`**?

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***).

Think 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,
    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.