CS111

Introduction to Computing Science

Recap

A while loop has this structure

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

- It starts with the keyword “while”
- Followed by a condition
- Followed by one or more statements.

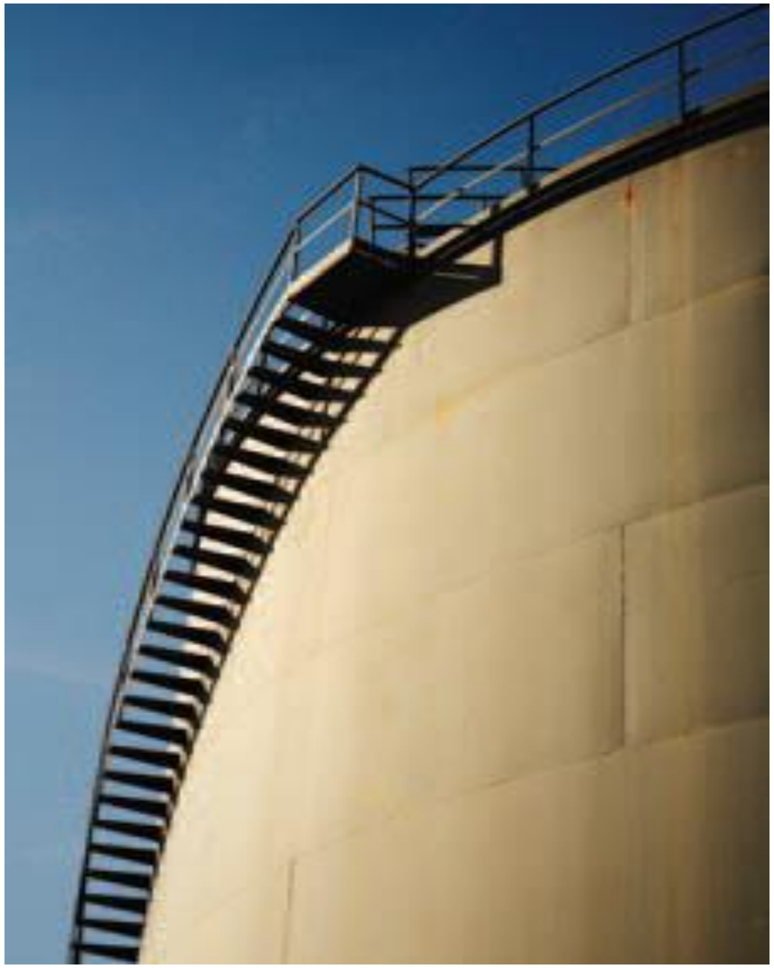
The *condition* is some kind of test (the same as in the **if** statement)

The statements are repeatedly executed while the condition is true.

The statements are also called the **body** of the while.

The loop stops when the condition is false.

The `for` Loop



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

To execute statements a certain number of times

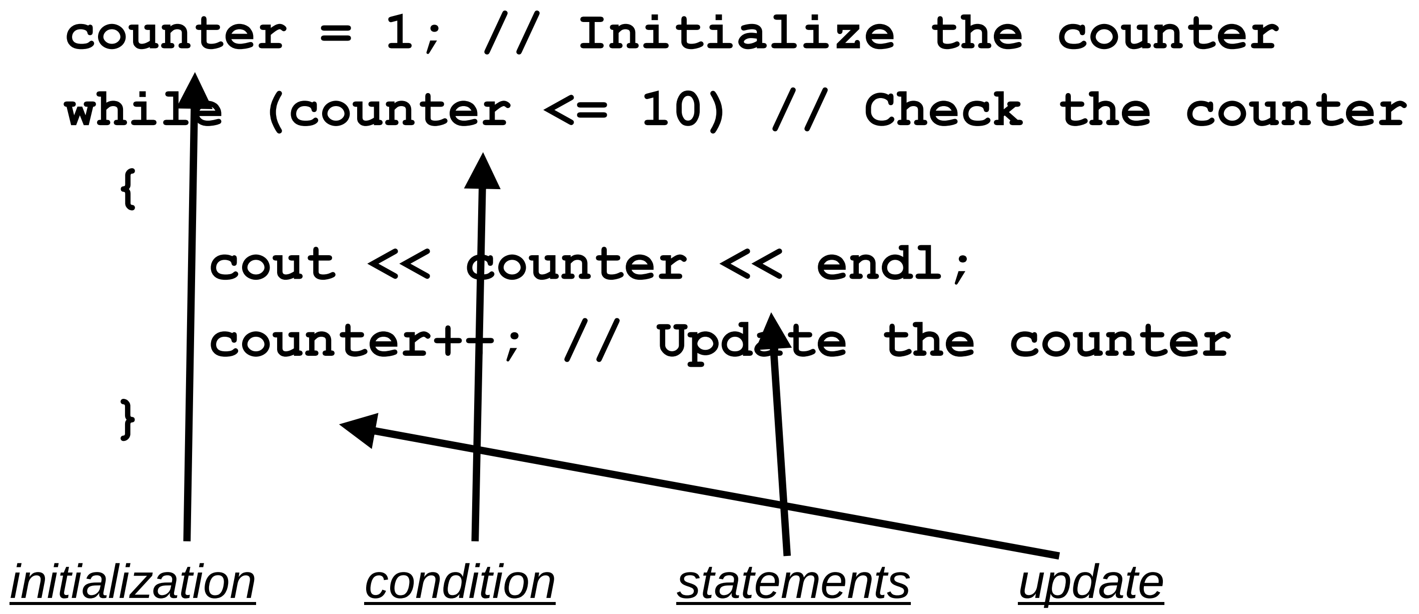
The `for` Loop

- Often you will need to execute a sequence of statements a given number of times.
- You could use a **`while`** loop for this.

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

The `for` Loop

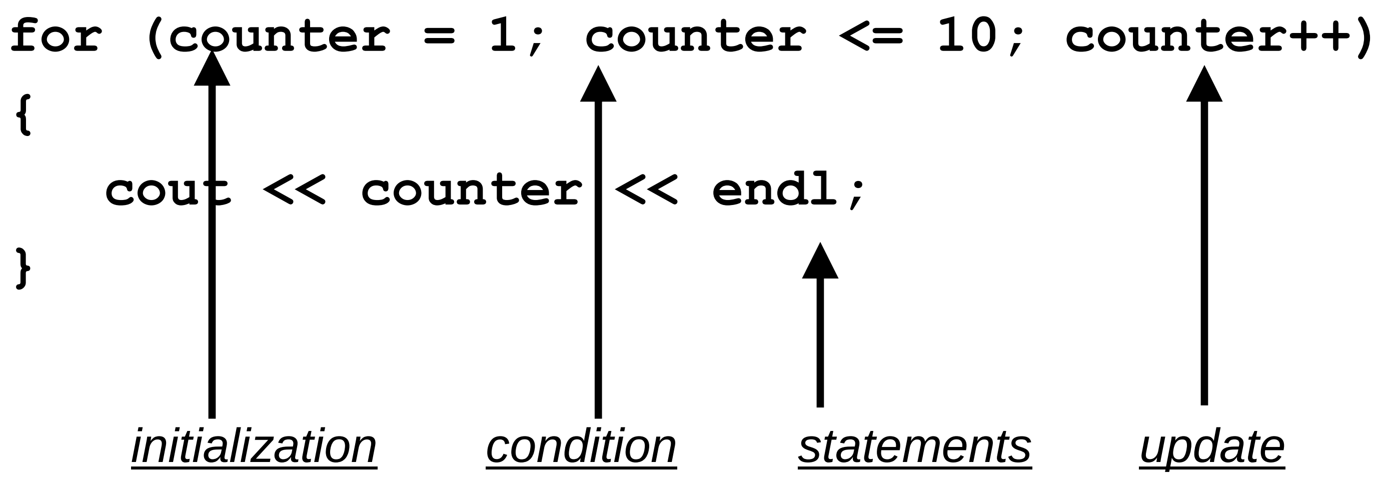
- The **`for`** loop is better than **`while`** for doing certain things
- Things that matter for the loop are all over the place.



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

The `for` Loop

- C++ has a statement custom made for this sort of processing: **`for`** loop
- The same now as a **`for`** loop.



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

The `for` Loop

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.

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

- The assignment `counter = 1;` is performed only once at the beginning.
- You can also declare and initialize the loop variable here.

The `for` Loop

The *condition* is code that tests to see if the loop is done. When this test is false, the **`for`** statement is over. We go on to the next statement after the closing bracket.

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

- The condition **`counter <= 10`** is checked before every loop.
- If the condition if false at the beginning, no loop will be taken.

The `for` Loop

The *statements* are repeatedly executed - until the condition is false. Thewse statements are also called the body of a **`for`** loop.

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

- If the condition is true this loop will print the value of **`counter`** to standard output.
- In this example it will print the numbers 1 to 10.

The `for` Loop

The *update* is code that is executed at the end of each loop, **before** the condition is checked**.** It causes the condition to eventually become false.

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

- The update **`counter++`** will increase the value of counter by 1 at the end of each loop.
- If this value exceeds 10 (**`counter>10`**), the **`for`** loop stops.

The `for` Loop

Some people call the **`for`** loop *count-controlled*.

- You initialize a counter.
- You check a counter.
- You update a counter.

In contrast, the **`while`** can be called an *event-controlled.*

- It executes until an event occurs, for example when the balance exceeds the target.

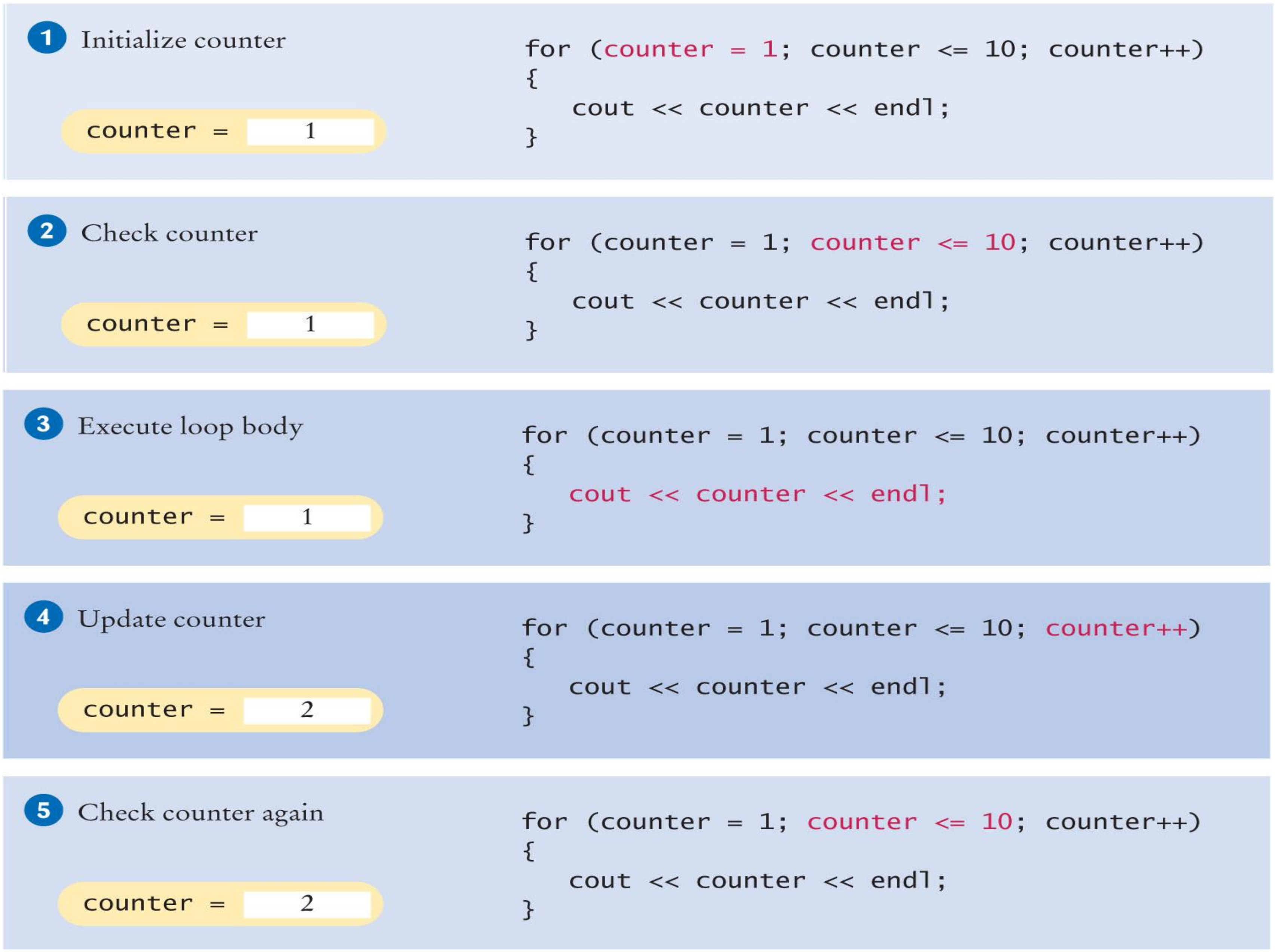
The `for` Loop

Another commonly-used term for a count-controlled loop is *definite*.

You know from the outset that the loop body will be executed a definite number of times—ten times in our example.

In contrast, event-controlled loops are called *indefinite*.

You do not know how many iterations it will take until the condition is false.



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

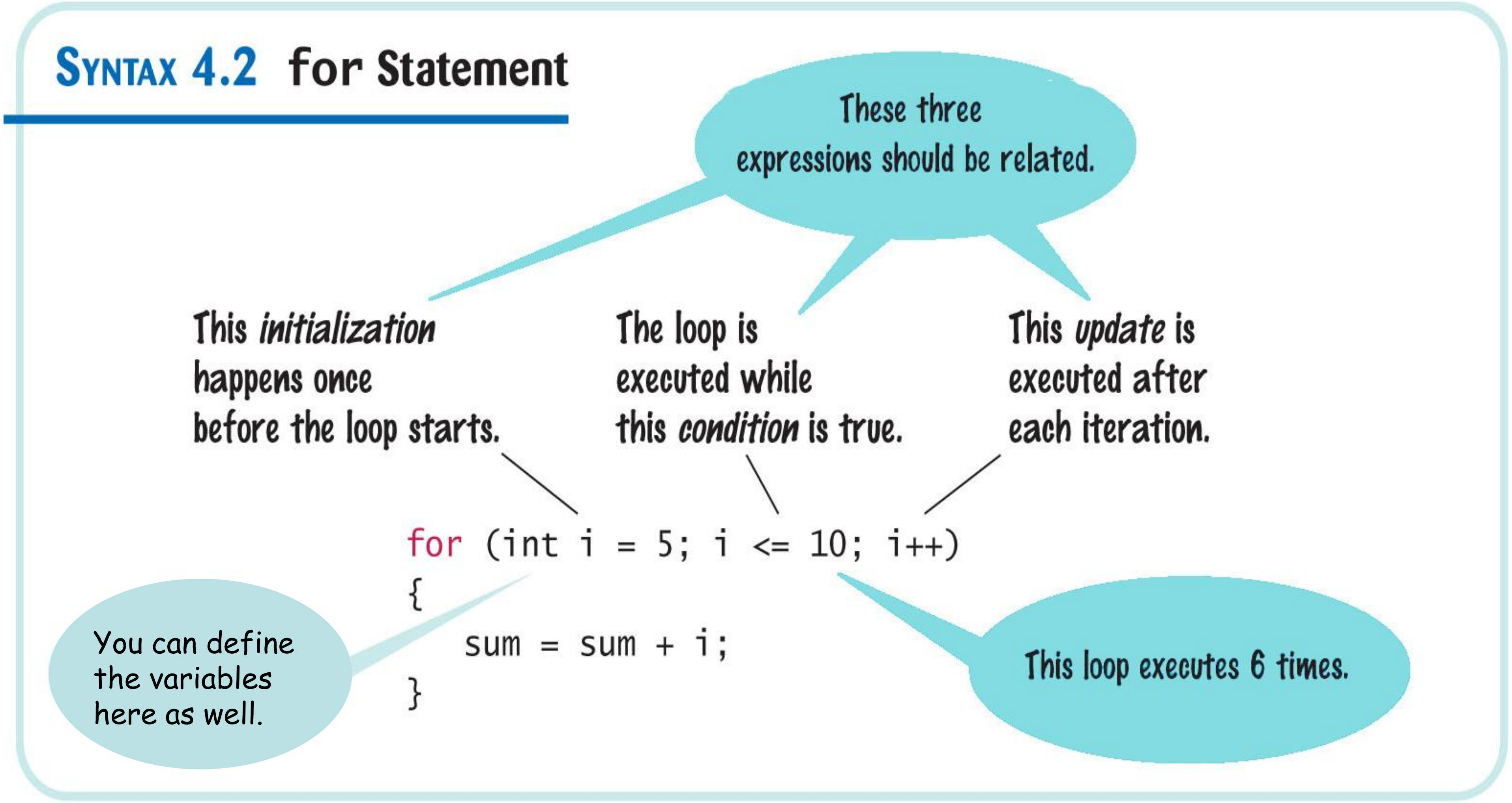
```
for (counter = 1; counter <= 10; counter++)
{
   cout << counter << endl;
}
```

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

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

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

The `for` Statement


SYNTAX 4.2 for Statement
These three expressions should be related.
This *initialization* happens once before the loop starts.
The loop is executed while this *condition* is true.
This *update* is executed after each iteration.
You can define the variables here as well.
This loop executes 6 times.

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

Scope of the Loop Variable

You can define (declare) the *loop variable* as part of the initialization

```
for(int counter = 5;…)
```

- When defined as part of the **`for`** statement cannot be used before or after the **`for`** statement .
- A **`for`** statement can use variables that were defined before the loop.
- In an earlier example, **`counter`** was defined before the loop – this works as well.
- However, many prefer to define loop variables in the i*nitialization*.

The `for` Can Count Up or Down

A **`for`** loop can count down instead of up:

```
for (int counter = 10; counter >= 0; counter--)…
```

The increment or decrement need not be in steps of 1

```
for (int counter = 0; counter <= 10; counter++)…
```

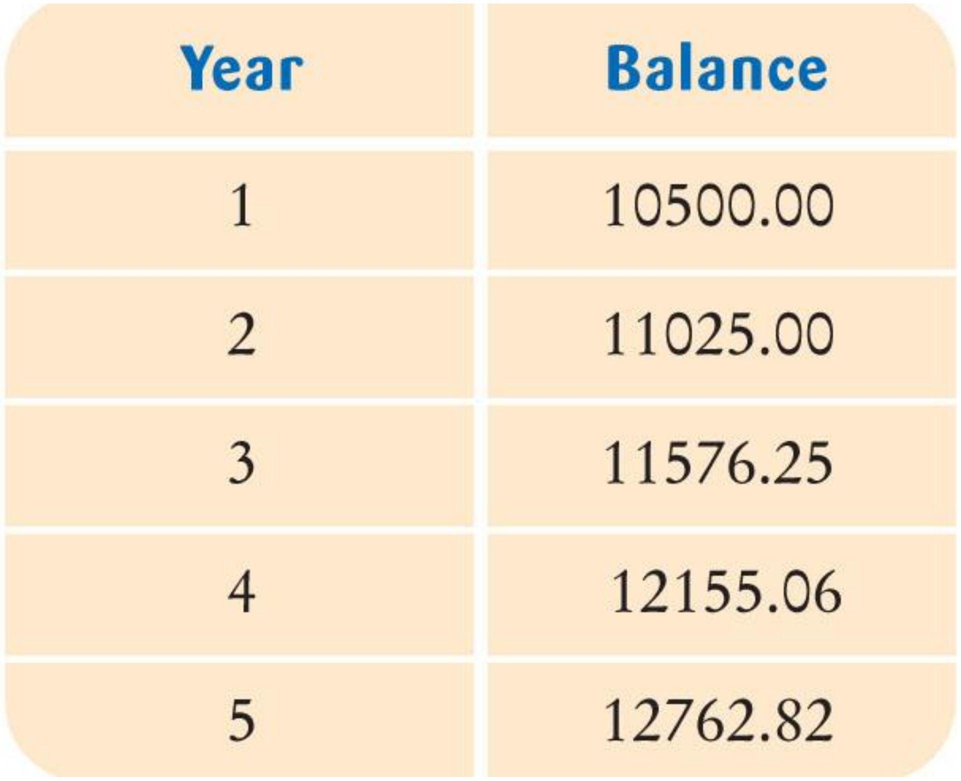
Notice that in these examples, the loop variable is defined **in** the *initialization* (where it really should be!).

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.

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

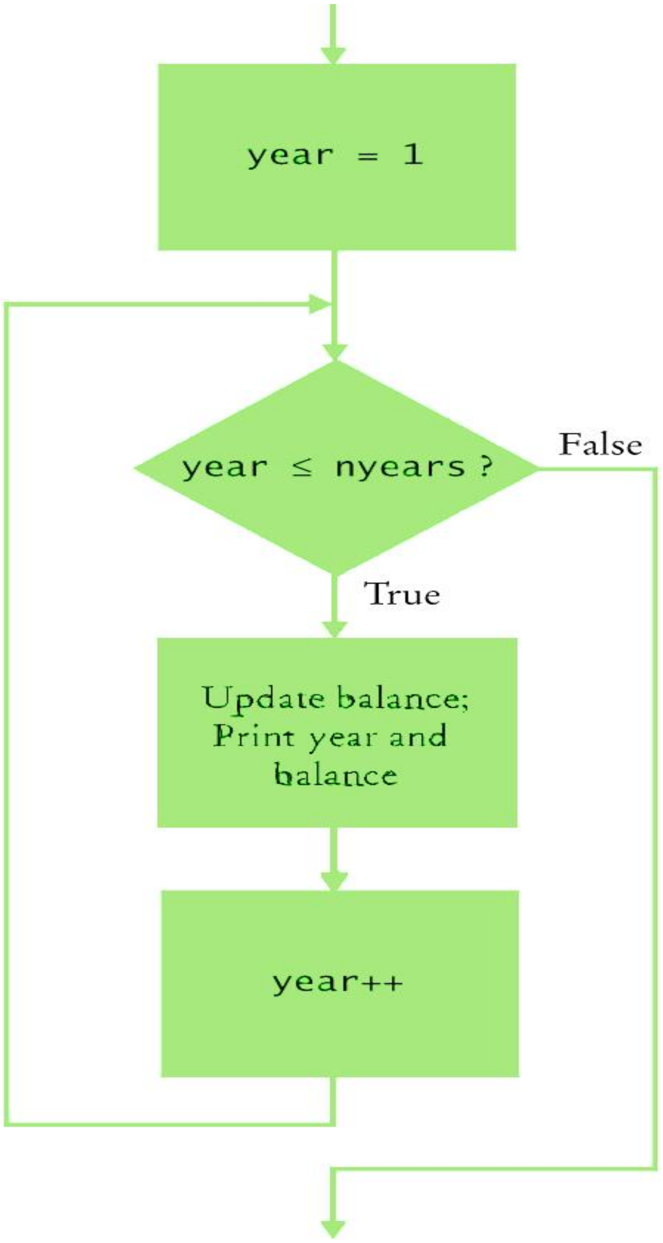
Solving a Problem with a `for` Statement

The pseudo-code

```
for year one to year five
      Update balance.
      Print year and balance.
```

The `for` Loop

Flowchart of
the investment
calculation
using
a for loop



Solving a Problem with a `for` Statement

Two statements should happen five times.

They are:

update balance

print year and balance

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

Solving a Problem with a `for` Statement

```
int main()
{
   const double RATE = 5;
   const double INITIAL_BALANCE = 10000;
   double balance = INITIAL_BALANCE;
   int nyears;
   cout << "Enter number of years: ";
   cin >> nyears;

   for (int year = 1; year <= nyears; year++)
   {
      balance = balance * (1 + RATE / 100);
      cout <<  year << ":\t" << balance << endl;
   }
   return 0;
}
```

Solving a Problem with a `for` Statement

A run of the program:

```
Enter number of years: 10
1:       10500
2:       11025
3:       11576.3
4:       12155.1
5:       12762.8
6:       13401
7:       14071
8:       14774.6
9:       15513.3
10:      16288.9
Press any key to continue . . .
```

Hand Tracing

```
   int main()
   {
      const double RATE = 5;
      const double INITIAL_BALANCE = 10000;
1     double balance = INITIAL_BALANCE;
2     int nyears;
3     cout << "Enter number of years: ";
4     cin >> nyears;

5     for (int year = 1;
6          year <= nyears;
7          year++)
      {
8        balance = balance * (1 + RATE / 100);
9        cout <<  year << ":\t" << balance << endl;
      }
10    return 0;
   }
```

Every part of the if gets their own line

Hand Tracing

```
   int main()
   {
      const double RATE = 5;
      const double INITIAL_BALANCE = 10000;
1     double balance = INITIAL_BALANCE;
2     int nyears;
3     cout << "Enter number of years: ";
4     cin >> nyears;

5     for (int year = 1;
6          year <= nyears;
7          year++)
      {
8        balance = balance * (1 + RATE / 100);
9        cout <<  year << ":\t" << balance << endl
      }
10    return 0;
   }
```

line			

Hand Tracing

```
   int main()
   {
      const double RATE = 5;
      const double INITIAL_BALANCE = 10000;
1     double balance = INITIAL_BALANCE;
2     int nyears;
3     cout << "Enter number of years: ";
4     cin >> nyears;

5     for (int year = 1;
6          year <= nyears;
7          year++)
      {
8        balance = balance * (1 + RATE / 100);
9        cout <<  year << ":\t" << balance << endl
      }
10    return 0;
   }
```

line	balance		
1	10000		

Hand Tracing

```
   int main()
   {
      const double RATE = 5;
      const double INITIAL_BALANCE = 10000;
1     double balance = INITIAL_BALANCE;
2     int nyears;
3     cout << "Enter number of years: ";
4     cin >> nyears;

5     for (int year = 1;
6          year <= nyears;
7          year++)
      {
8        balance = balance * (1 + RATE / 100);
9        cout <<  year << ":\t" << balance << endl
      }
10    return 0;
   }
```

input is 2

line	balance	nyears	
1	10000		
2	"		
3	"		
4	"	2	

Hand Tracing

```
   int main()
   {
      const double RATE = 5;
      const double INITIAL_BALANCE = 10000;
1     double balance = INITIAL_BALANCE;
2     int nyears;
3     cout << "Enter number of years: ";
4     cin >> nyears;

5     for (int year = 1;
6           year <= nyears;
7           year++)
      {
8        balance = balance * (1 + RATE / 100);
9        cout <<  year << ":\t" << balance << endl
      }
10    return 0;
   }
```

condition is true

continue here

line	balance	nyears	year
1	10000		
2	"		
3	"		
4	"	2	
5	"	"	1
6	"	"	"

Hand Tracing

```
   int main()
   {
      const double RATE = 5;
      const double INITIAL_BALANCE = 10000;
1     double balance = INITIAL_BALANCE;
2     int nyears;
3     cout << "Enter number of years: ";
4     cin >> nyears;

5     for (int year = 1;
6          year <= nyears;
7          year++)
      {
8        balance = balance * (1 + RATE / 100);
9        cout <<  year << ":\t" << balance << endl
      }
10    return 0;
   }
```

condition is true

line	balance	nyears	year
1	10000		
2	"		
3	"		
4	"	2	
5	"	"	1
6	"	"	"
8	10500	"	"
9	"	"	"
7	"	"	2
6	"	"	"

Hand Tracing

```
   int main()
   {
      const double RATE = 5;
      const double INITIAL_BALANCE = 10000;
1     double balance = INITIAL_BALANCE;
2     int nyears;
3     cout << "Enter number of years: ";
4     cin >> nyears;

5     for (int year = 1;
6          year <= nyears;
7          year++)
      {
8        balance = balance * (1 + RATE / 100);
9        cout <<  year << ":\t" << balance << endl
      }
10    return 0;
   }
```

condition is false

line	balance	nyears	year
1	10000		
2	"		
3	"		
4	"	2	
5	"	"	1
6	"	"	"
8	10500	"	"
9	"	"	"
7	"	"	2
6	"	"	"
8	11025	"	"
9	"	"	"
7	"	"	3
6	"	"	"
10	"	"	"

Common Error- Confusing 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.
- You can write this:

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

- It compiles, and it works, but will confuse everyone, including yourself.

Common Error- Confusing 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.

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++)…
```

Know Your Bounds – Symmetric vs. Asymmetric

- 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.

How Many Times Was That?

Fence arithmetic

Don’t forget to count the first (or last)
“post number” that a loop variable takes on.

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.

Fence Arithmetic – Counting Iterations

Counting is <u>easier</u> for loops with *asymmetric* bounds.

The loop

```
for (i = a; i < b; i++)…
```

executes the statements **`(b – a)`** times
and when a is 0.

Fence Arithmetic – Counting Iterations

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`**.

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.

The `do` Loop

The while loop’s condition test is the first thing
that occurs in its execution.

The **`do`** loop (or **`do-while`** loop) has its condition tested only after at least one execution of the statements.

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

The `do` Loop

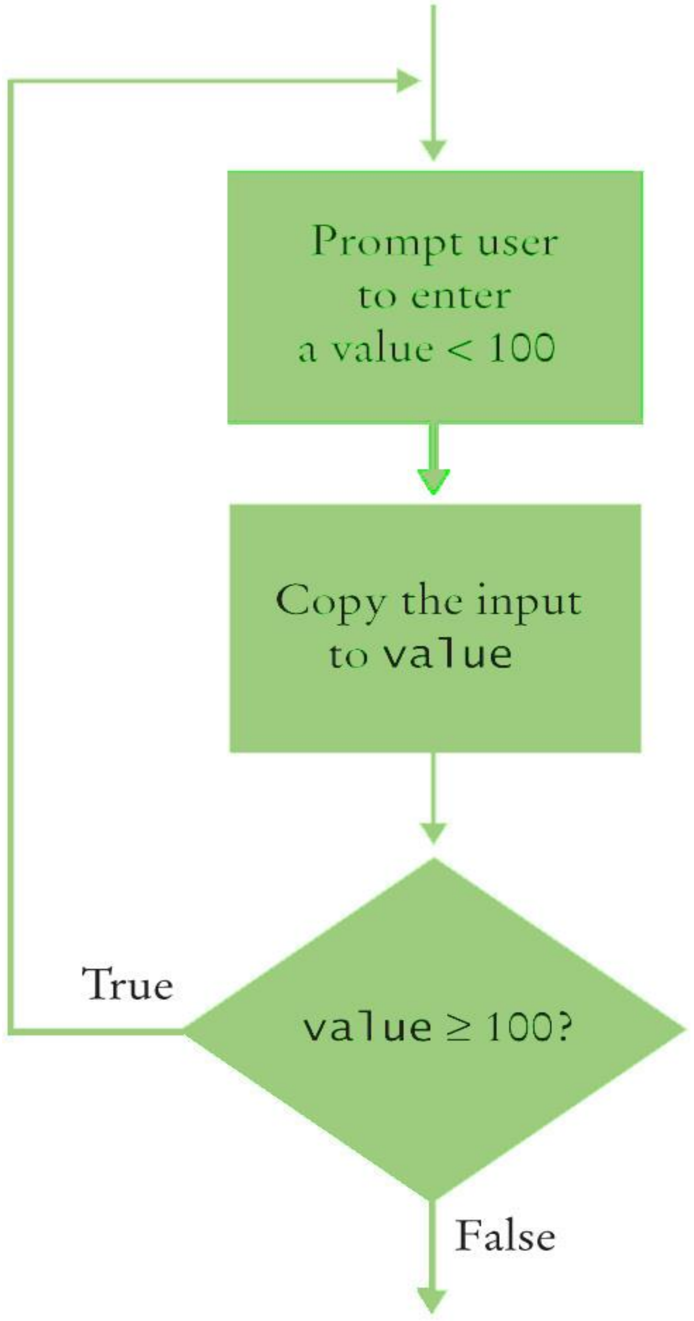
This means that the do loop should be used only when the statements must be executed before there is any knowledge of the condition.

This also means that in practice the `do` loop is the least used loop.

The do Loop

- What problems require something to have happened before the testing in a loop?
- Getting valid user input is often cited.

- Here is the flowchart for the problem in which the user is supposed to enter a value less than 100 and processing must not continue until they do.



The `do` Loop

- Here is the code:

```
int value;
do
{
   cout << "Enter a value < 100: ";
   cin >> value;
}
while (value >= 100);
```

- In this form, the user sees the same prompt each time until the enter valid input.

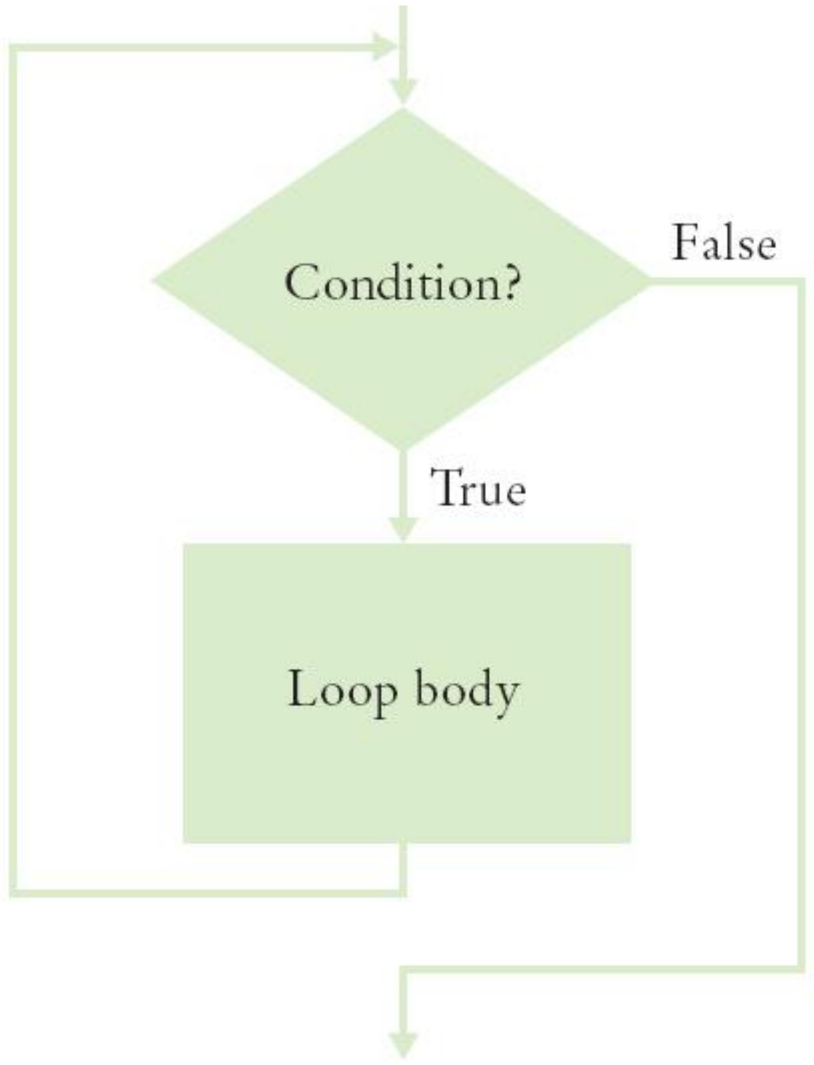
The `do` Loop

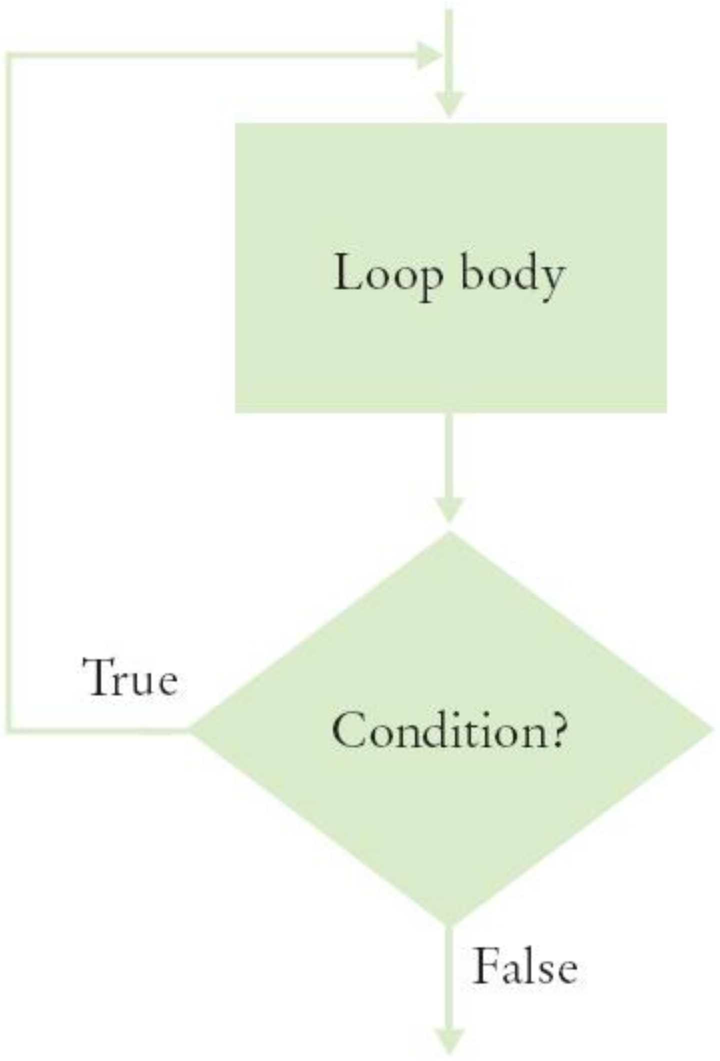
In order to have a different, “error” prompt that the user sees only on *invalid* input, the initial prompt and input would be before a while loop:

```
int value;
cout << "Enter a value < 100:";
while (value >= 100);
{
   cout << "Sorry, that is larger than 100\n"
        << "Try again: ";
   cin >> value;
}
```

```
Notice that nothing
happens if the user
gives valid input
on the first
attempt. Good.
```

Flowcharts for the `while` Loop and the `do` Loop





Processing Input – When and/or How to Stop?

- We need to know, when getting input from a user, when they are done.
- One method is to hire a sentinel (as shown)

  or more correctly choose a *value* whose meaning is STOP!

- As long as there is a known range of valid data points, we can use a value not in it.

Processing Input – When and/or How to Stop?

We will write code to calculate the average of some salary values input by the user.

How many will there be?

- That is the problem. We can't know.
- But we can use a *sentinel value*, as long as we tell the user to use it, to tell us when they are done.
- Since salaries are never negative, we can safely choose -1 as our sentinel value.

Processing Input – When and/or How to Stop?

- In order to have a value to test, we will need to get the first input before the loop.
- The loop statements will process each non-sentinel value, and then get the next input.
- For averages we need the total sum, and the total number of inputs.

Pseudo code

```
ask for input
while input is not negative
    update totals
    ask for input
compute average
```

The Complete Salary Average Program

```
int main()
{
   double sum = 0;
   int count = 0;
   double salary = 0;
   cout << "Enter salaries, -1 to finish: ";
   cin >> salary;
   while (salary != -1)
   {
        sum = sum + salary;
        count++;
        cin >> salary;
    }
   cout << “The average is: " << sum/count << endl;
   return 0;
}
```

Quiz

- You need add the scores of 10 students. Which type of loop do you use?

Quiz

- You want to add student scores until the user enters a negative score. Which type of loop do you use?

Quiz

- What does the condition is a while loop say? Whether to stop the loop. Or whether to continue?

Quiz

- How often does initialisation in a for-loop take place, and when?

Quiz

- How often is the condition in a for-loop take checked, and when?

Quiz

- How often is the update in a for-loop take performed, and when?

Quiz

- At the end of an iteration in a for loop, do you update first and then check the condition, or do you check first and then update?

Quiz

- In the first statement after a while-loop, is the condition true or false.

```
while (condition){
 do something
}

next statement;
```

Quiz

- What value will be printed after the while-loop?

```
while (counter<=10){
  counter++;
}

cout << counter << endl;
```

Quiz

- In the first statement after a for-loop, is the condition true or false.

```
for (initialisation;condition;update){
 do something
}

next statement;
```

Quiz

- What value will be printed after the for-loop?

```
for (int counter = 0; counter < 10; counter++){
  cout << 2*counter << endl;
}

cout << counter << endl;
```