

EN47/COMP9

Exploring Computer Science

Lecture 4

Controlling Program Flow:
Conditionals and Iteration

Outline

Conditional flow: `if-else` statements

Iterations: `while` and `for` loops

Finding the fake coin

I give you three coins.

One of them is fake.



The fake one is heavier than the others.

You have a scale.

What is the fewest number of weighings
you can use to find the fake?

Finding the fake coin

Weigh two coins against each other:

- If the left one is heavier, it's the fake.
- If the right one is heavier, it's the fake.
- Otherwise, the remaining coin is the fake



Can we write a C++ program for this algorithm?

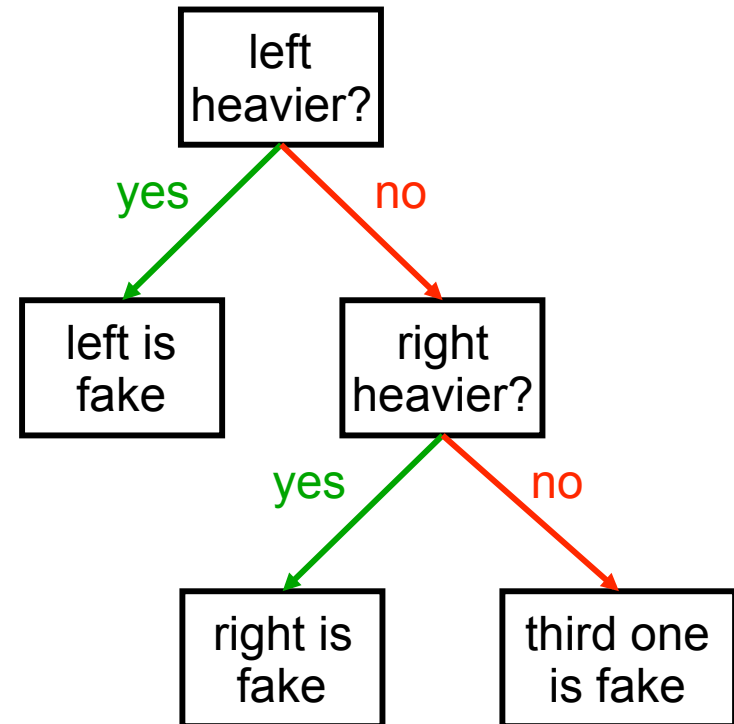
Control flow of the weighings

Here's our algorithm:

- *control* of the process...
- *flows* from box to box.

The algorithm is clearly stated and deterministic.

The computer should be able to do it!



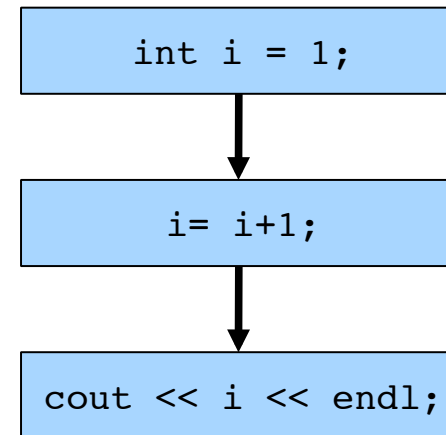
Control flow

Control flow is the order in which statements are executed

Up to now, our control flow has been *sequential*:

- The next statement executed is the next one that appears, in order, in the C++ program

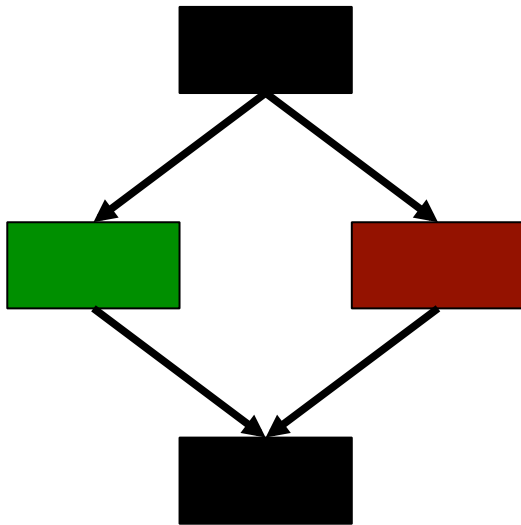
```
{  
  int i = 1;  
  i = i + 1;  
  cout << i << endl;  
}
```



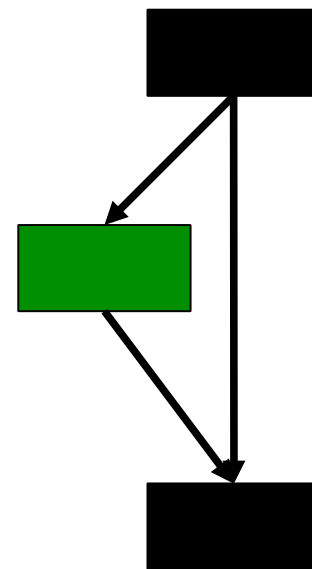
Notice the `{ }`. More about this later...

Conditional control flow

Choosing which of the two statements to execute before continuing



Deciding whether or not to skip a statement before continuing



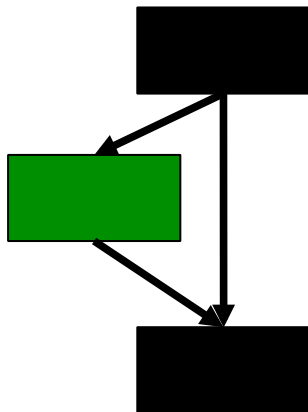
Conditional execution

Conditional statements allow the computer to choose an execution path depending on the value of a variable or expression.

- If the withdrawal is more than the balance, then print an error.
- If today is my birthday, then add one to my age.
- If it's a 9:30 class, then prop your eyelids open; otherwise, gnaw on your arm while you wait for lunch.

The basic `if` statement

Execute the code within the statement only if a particular condition is true.



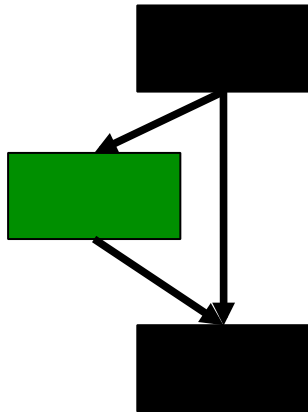
```
if (condition)  
statement;
```

The statement is executed if the condition is `true`.

Otherwise, the statement is skipped.

The basic if statement

Execute the code within the statement only if a particular condition is true.



Examples:

```
if (withdrawalAmount > balance)
    cout << "Not enough money" << endl;
```

```
if (x < 100)
    x = x + 1;
```

```
if (temperature > 98.6)
    cout << "You have a fever." << endl;
    cout << "Go see a doctor!" << endl;
```

Something is different with this last one...

Compound statements

Group statements with { } so that they are treated as a single statement

Indicates sequential control flow

Also called a **block**

```
{  
    statement1;  
    statement2;  
    ...  
}
```

Blocks are necessary

To perform multiple statements conditionally, use compound statements:

```
if (condition) {  
    statement 1;  
    statement 2;  
  
    ...  
}
```

Example:

```
if (temperature > 98.6) {  
    cout << "You have a fever." << endl;  
    cout << "Go see a doctor!" << endl;  
}
```

- If condition is `true`, all statements between the braces are executed
- Otherwise, all the statements between the braces are skipped

Conditions

Logical expressions (also called **Boolean** expressions)

Made up of variables, constants, arithmetic expressions,
and the **relational operators**

Math symbols :	<	≤	>	≥	=	≠
in C++ :	<	<=	>	>=	==	!=

Equality (==) and assignment (=) are different concepts!

The value of a conditional expression

True or false (1 bit of information)

Under the hood of C++, that value is an integer:

- 0 is interpreted as false
- Any non-zero integer value (e.g. 1) is interpreted as true

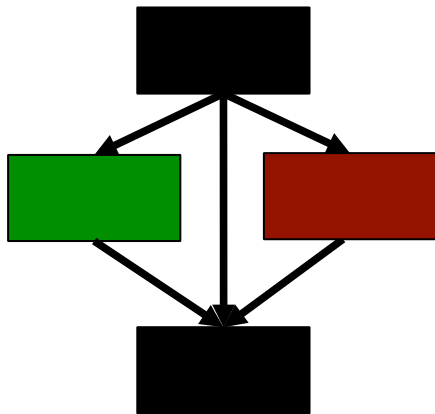
The value of a conditional expression

What are the values of these expressions? (Suppose x is 10.)

- $(12 > 5)$
- $(7 \leq 5)$
- $(x == 10)$
- $(x == 8)$
- $(x != 8)$

The `if-else` statement

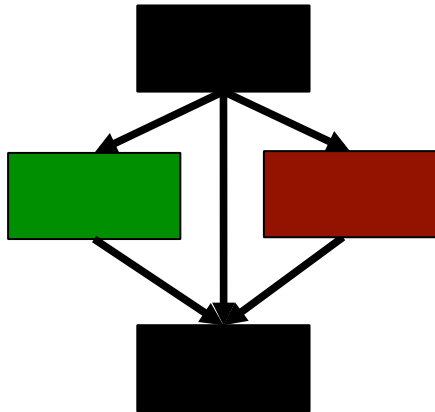
Execute one block of code if the condition is true and another if it is false.



```
if (condition) {  
    statement1;  
}  
else {  
    statement2;  
}
```

The `if-else` statement

Execute one block of code if the condition is true and another if it is false.

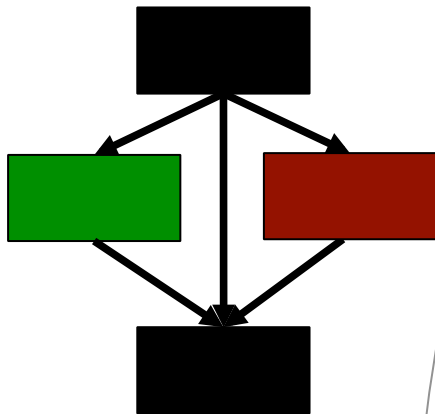


Example:

```
if (withdrawalAmount <= balance) {  
    balance = balance - withdrawalAmount;  
    cout << "Withdrawal done." << endl;  
}  
else {  
    cout << "Not enough money" << endl;  
}  
cout << "Thank you for banking today.";
```

The `if-else` statement

Execute one block of code if the condition is true and another if it is false.



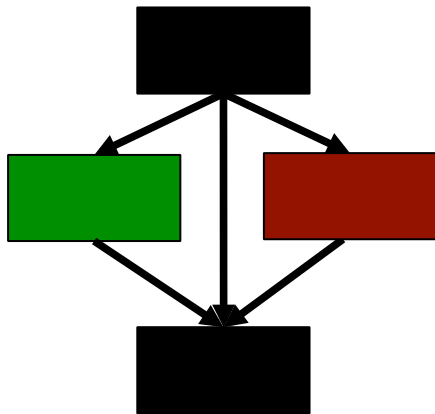
Example:

```
if (withdrawalAmount <= balance) {  
    balance = balance - withdrawalAmount;  
    cout << "Withdrawal done." << endl;  
}  
else {  
    cout << "Not enough money" << endl;  
}  
cout << "Thank you for banking today.";
```

keywords

The `if-else` statement

Execute one block of code if the condition is true and another if it is false.



Example:

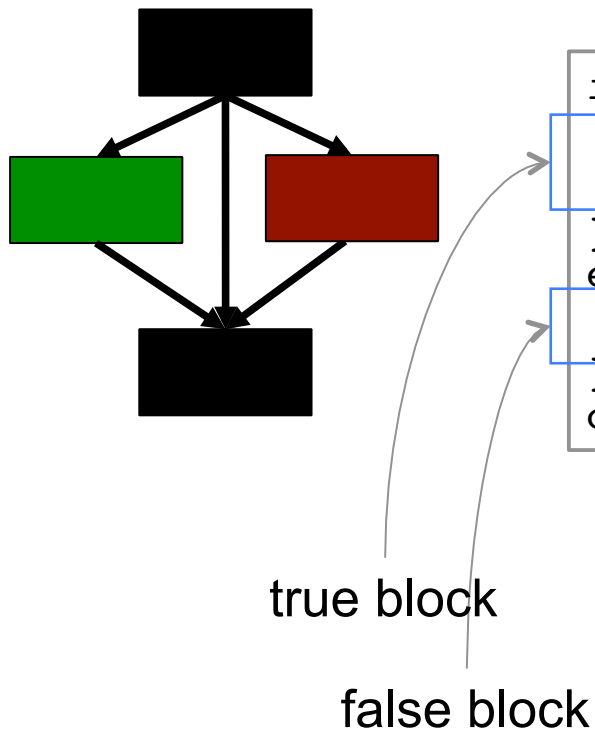
conditional expression

```
if (withdrawalAmount <= balance) {  
    balance = balance - withdrawalAmount;  
    cout << "Withdrawal done." << endl;  
}  
else {  
    cout << "Not enough money" << endl;  
}  
cout << "Thank you for banking today.";
```

The `if-else` statement

Execute one block of code if the condition is true and another if it is false.

Example:



```
if (withdrawalAmount <= balance) {  
    balance = balance - withdrawalAmount;  
    cout << "Withdrawal done." << endl;  
}  
else {  
    cout << "Not enough money" << endl;  
}  
cout << "Thank you for banking today.";
```

Example: Computing absolute value

Problem: Compute the absolute value of x

Put the result in variable abs

```
if (x >= 0) {  
    abs = x;  
}  
else {  
    abs = -x;  
}
```

```
if (x >= 0) {  
    abs = x;  
}  
if (x < 0) {  
    abs = -x;  
}
```

```
abs = x;  
if (x < 0) {  
    abs = -x;  
}
```

Spot the differences? Which code is correct?

Chained if-else statements

Can get arbitrarily complex...

```
string size = "unknown";
if (area > 10) {
    size = "big";
}
else if (area > 5) {
    size = "medium";
}
else {
    size = "small";
}
```

```
string size = "unknown";
if (area > 10) {
    size = "big";
}
else {
    if (area > 5) {
        size = "medium";
    }
    else {
        size = "small";
    }
}
```

Sudoku: Print the grid

Look at content of first position

Print content

Look at next position

Print content

Look at next position

Print content

...

	8					3		2
			7		1			8
		4	9	3			1	
	6	3						
5	9						3	4
						8	9	
	4			1	2	7		
6			4		7			
8		7					5	

Sudoku: Print the grid

We want to repeat the same process for each position

In other words, we want to **loop** over the positions

Another algorithm:

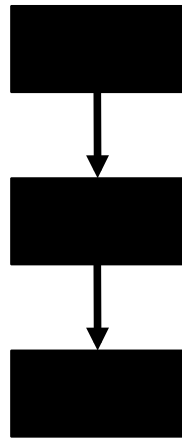
- Loop over lines
- For each line, loop over columns

	8					3		2
			7		1			8
		4	9	3			1	
	6	3						
5	9						3	4
						8	9	
	4			1	2	7		
6			4		7			
8		7					5	

Another form of control flow

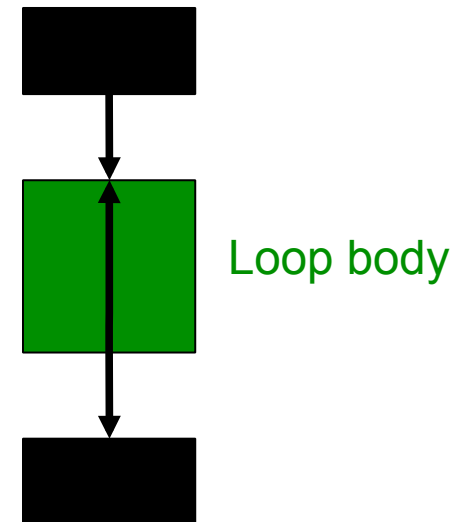
Sequential

- Statements execute in order



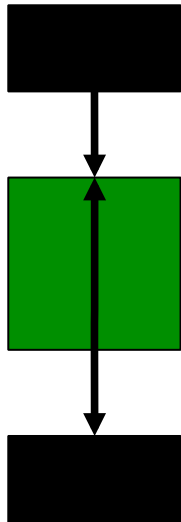
Loop

- Repeat a block of code



The `while` loop

Execute a block of code repeatedly while a condition is true.



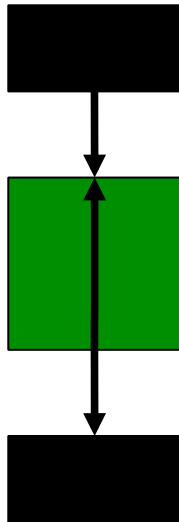
```
while (condition) {  
    statement 1;  
    statement 2;  
    ...  
}
```

The while loop

Execute a block of code repeatedly while a condition is true.

Example:

```
while (withdrawalAmount <= balance) {  
    balance = balance - withdrawalAmount;  
    cout << "Withdrawal done." << endl;  
}  
cout << "No money left." << endl;
```

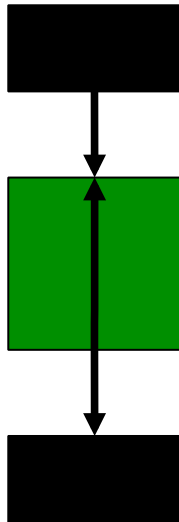


The while loop

Execute a block of code repeatedly while a condition is true.

Example:

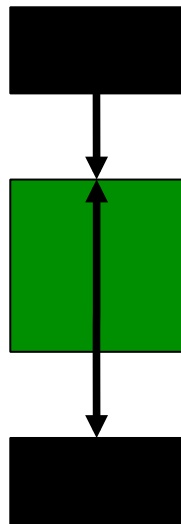
```
while (withdrawalAmount <= balance) {  
    balance = balance - withdrawalAmount;  
    cout << "Withdrawal done." << endl;  
}  
cout << "No money left." << endl;
```



keyword

The while loop

Execute a block of code repeatedly while a condition is true.



Example:

conditional expression

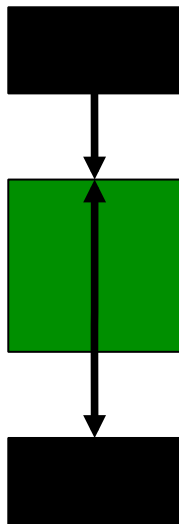
```
while (withdrawalAmount <= balance) {  
    balance = balance - withdrawalAmount;  
    cout << "Withdrawal done." << endl;  
}  
cout << "No money left." << endl;
```

The while loop

Execute one block of code if the condition is true and another if it is false.

Example:

```
while (withdrawalAmount <= balance) {  
    balance = balance - withdrawalAmount;  
    cout << "Withdrawal done." << endl;  
}  
cout << "No money left." << endl;
```



loop body

A loopless problem (?)

Problem: Add four numbers entered at the keyboard.

```
int sum;
int x1, x2, x3, x4;

cout << "Enter 4 numbers: ";
cin >> x1 >> x2 >> x3 >> x4;

sum = x1 + x2 + x3 + x4;
```

This code works perfectly (and sequentially).

What if we had 14, or 40, or 400 numbers?

Writing loops == Generalizing

Problem: Add four numbers entered at the keyboard

```
int sum, x;
sum = 0;
cout << "Enter 4 numbers: ";

cin >> x;
sum = sum + x;

cin >> x;
sum = sum + x;

cin >> x;
sum = sum + x;

cin >> x;
sum = sum + x;
```

```
int sum, x;
int count;

sum = 0;
count = 1;
cout << "Enter 4 numbers: ";

while (count <= 4) {
    cin >> x;
    sum = sum + x;
    count = count + 1;
}
```

The for loop

Execute a block of code a specified number of times.

```
for (initialization; condition; update) {  
    statement1;  
    statement 2;  
    ...  
}
```

The for loop

Execute a block of code a specified number of times.

```
int sum, x;
int count;

sum = 0;

cout << "Enter 4 numbers: ";

for (count = 1; count <= 4; count = count + 1) {
    cin >> x;
    sum = sum + x;
}

cout << sum << endl;
```

The for loop

Execute a block of code a specified number of times.

```
int sum, x;
int count;

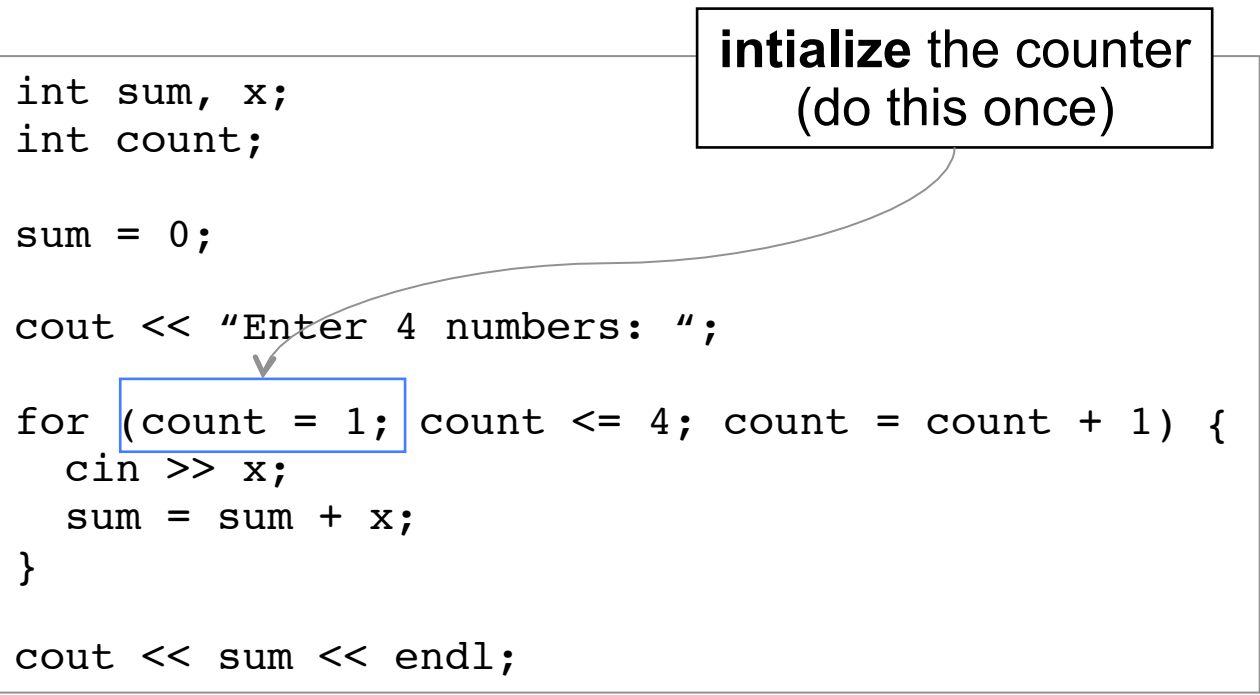
sum = 0;

cout << "Enter 4 numbers: ";

for (count = 1; count <= 4; count = count + 1) {
    cin >> x;
    sum = sum + x;
}

cout << sum << endl;
```

intialize the counter
(do this once)



The for loop

Execute a block of code a specified number of times.

```
int sum, x;
int count;

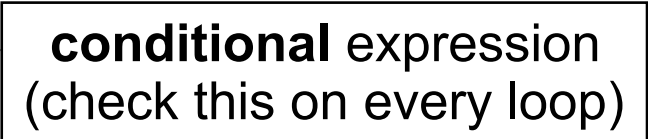
sum = 0;

cout << "Enter 4 numbers: ";

for (count = 1; count <= 4; count = count + 1) {
    cin >> x;
    sum = sum + x;
}

cout << sum << endl;
```

conditional expression
(check this on every loop)



The for loop

Execute a block of code a specified number of times.

```
int sum, x;
int count;

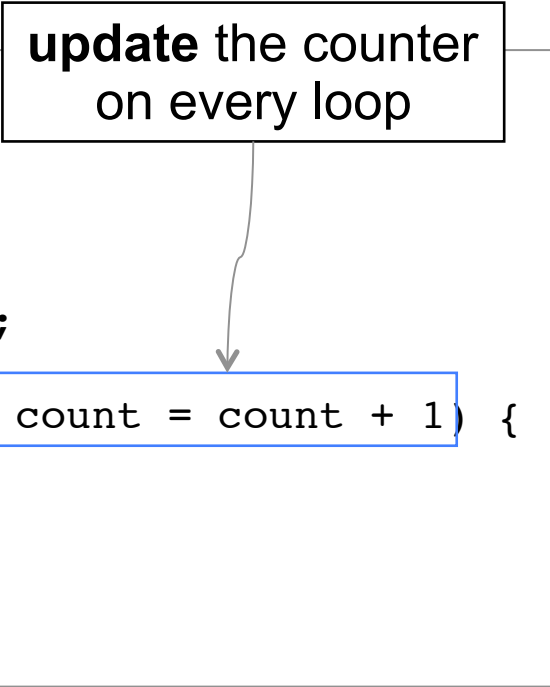
sum = 0;

cout << "Enter 4 numbers: ";

for (count = 1; count <= 4; count = count + 1) {
    cin >> x;
    sum = sum + x;
}

cout << sum << endl;
```

**update the counter
on every loop**



Counting in for Loops

```
// Print n asterisks

for (count = 1; count <= n; count = count + 1) {
    cout << "*";
}

// A different style of counting:

for (count = 0; count < n; count = count + 1) {
    cout << "*";
}
```

Counting up or down by 1

Increment (increase) and **decrement** (decrease) by 1:

C/C++ operators

- increment: ++
- decrement: --
- This is where C++'s name comes from:
 - C++ is an “increment” over C

```
// Print n asterisks  
  
for (count = 0; count < n; count++) {  
    cout << "*" ;  
}
```

count++ means count = count+1

while VS. for

```
int sum, x;
int count;

sum = 0;
count=1;
cout << "Enter 4 numbers: ";

while (count <= 4) {
    cin >> x;
    sum = sum + x;
    count=count+1;
}

cout << sum << endl;
```

```
int sum, x;
int count;

sum = 0;

cout << "Enter 4 numbers: ";

for (count = 1; count <= 4; count = count + 1) {
    cin >> x;
    sum = sum + x;
}

cout << sum << endl;
```

while VS. for

These two loops mean exactly the same thing

Any `for` loop can be written as a `while` loop

```
while (condition) {  
    statement 1;  
    statement 2;  
  
    ...  
    update;  
}
```

```
for (initialization; condition; update) {  
    statement 1;  
    statement 2;  
  
    ...  
}
```

Keep in mind...

Always use integers as loop counters

Make sure you have the right ' ; ' in the right place

Always use blocks { } to avoid confusions

Make sure the condition will be met at some point

- Otherwise, **infinite loop!**

Summary

Conditional flow

- `if-else` statements

Iterations

- General pattern: initialize, test, do stuff, repeat...
- `while` and `for` are equally general in C/C++
- Use `for` when initialize/test/update are closely related and simple, especially when counting

Lab 2

Problem: Find the average size of n grids

Specifications:

- User inputs n , then numlines and numcols of each grid
- If $n=0$, then print “No grid size was computed”
- If $n>0$, then for each grid, print its size and the average grid size so far

Lab 2

The final output of your program should look like:

```
How many grids? 3

How many lines in grid 1? 10
How many columns? 3
The grid size is 30 cells.

How many lines in grid 2? 5
How many columns? 3
The grid size is 15 cells.
The average grid size so far is 22.5 cells.

How many lines in grid 3? 9
How many columns? 7
The grid size is 63 cells.
The average grid size so far is 36 cells.

The sizes of 3 grids were computed.
```

If no grid size is calculated, the output should look like:

```
How many grids? 0
No grid size was computed.
```