

COMP 10
EXPLORING COMPUTER SCIENCE

Lecture 3
Controlling Program Flow:
Conditionals and Iteration

A PUZZLE



One of the coins is a fake

The fake one is heavier than the others

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

A PUZZLE



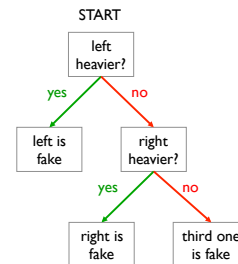
Weigh two coins against each other:

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

A WEIGHING ALGORITHM



The algorithm is clearly stated and deterministic.

The computer should be able to do it!

CONTROL FLOW

The order in which statements are executed

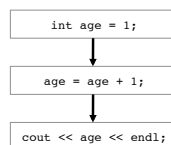
Up to now, our control flow has been sequential:

Code:

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

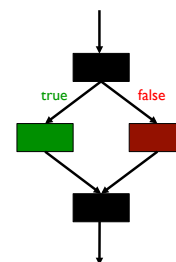
Notice the { }. More about this later...

Control Flow:



CONDITIONAL CONTROL FLOW

Choose which statement to execute before continuing



CONDITIONAL STATEMENTS

Choose an **execution path** depending on the value of a variable or expression

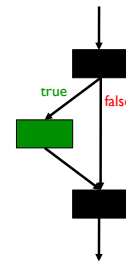
If today is my birthday, then add one to my age.

If the withdrawal is more than the balance, then print an error.

If it's 9:30, then wake up for class; otherwise, hit 'Snooze'.

THE `if` STATEMENT

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

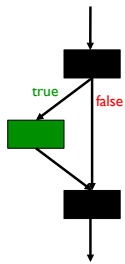


```
if (condition is true)
    statement;
```

The statement is executed if the condition is `true`.
Otherwise, the statement is skipped.

THE `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 `{ }` to enforce sequential control flow

Also called a **block**

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

COMPOUND `if` STATEMENTS

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

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

TESTING, TESTING

Logical expressions (also called Boolean expressions)

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

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

Math symbols : $< \leq > \geq = \neq$
in C++ : `< <= > >= == !=`

Note: 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)

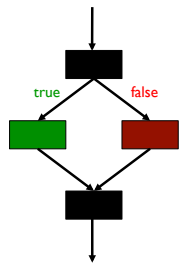
(7 <= 5)

(x == 10)

(x == 8)

THE if-else STATEMENT

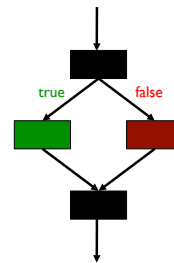
Execute one block of code if the condition is true and another if it is false.



```
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.

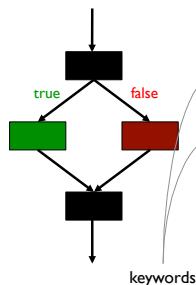


Example:

```
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.



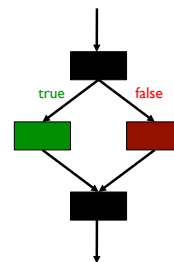
Example:

```
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.



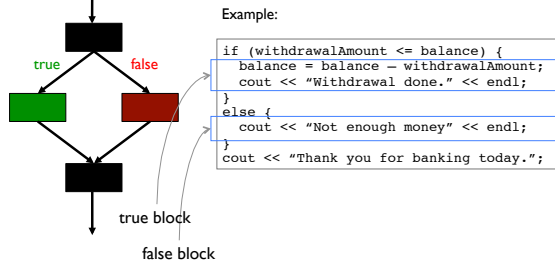
Example:

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

conditional expression

THE if-else STATEMENT

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



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:

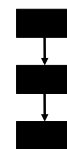
Loop over lines
For each line, loop over columns

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

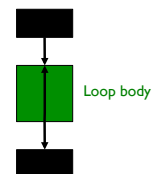
Sequential

Statements execute in order



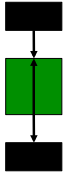
Loop

Repeat a block of code



THE while LOOP

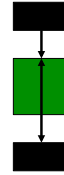
Execute a block of code repeatedly while a condition is true.



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

THE while LOOP

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

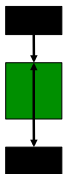


Example:

```
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.



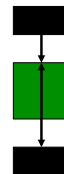
Example:

```
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.



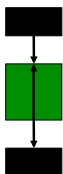
Example:

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

conditional expression

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) {
    statement 1;
    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;
```

initialize 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:

```
// 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

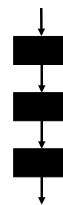
Make sure you have the right ' ; ' in the right place

Always use blocks { } to avoid confusions

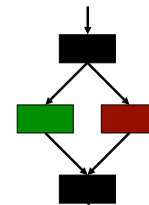
Make sure the condition will be met at some point
Otherwise, **infinite loop!**

Summary

Sequential

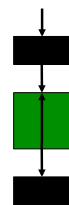


Conditional



if, else

Loop



while, for

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 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? 2
How many columns? 3
The grid size is 6 cells.
The average grid size so far is 18 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.
```