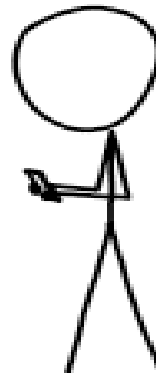
Strings & Branching



WHY I TRY NOT TO BE PEDANTIC ABOUT CONDITIONALS.

bool

bool - either true or false

You have the common math comparisons:

> (greater than), e.g. 7 > 2.5 is true

== (equals), e.g. 5 == 4 is false

<= (less than or eq), e.g. 1 <= 1 is true

If you cout this, “false” will be 0
and “true” will be 1 (anything non-zero is T)

if statement

Code inside an if statement is only run if the condition is true.

Need parenthesis (no semi-colon)

```
12      if(guess == random0to9)
13      {
14          cout << "Correct, here is a cookie!\n";
15      }
```

Indent

(See: ifElse.cpp)

boolean values

ints will automatically be converted to bool, which can cause errors:

```
int x = 2;
if( !  x>5  )  will be false
```

Why?

boolean values

ints will automatically be converted to bool, which can cause errors:

int x = 2;

if(! x>5) will be false

Why?

A: order of operations will do the unary operator first (the '!')

if (! x>5) will become if ((!2) > 5)

... if ((!true) > 5) ... if (false > 5) ... if (0 > 5)

if/else statement

Immediately after an if statement,
you can make an else statement

If the “if statement” does not run,
then the else statement will

If you do not surround your code with braces
only one line will be in the if (and/or else)
statement

Logical operators

These are all the operators that result in a bool:

> (greater than), e.g. 7 > 2.5 is true

== (equals), e.g. 5 == 4 is false

< (less than), e.g. 1 < 1 is false

>= (greater than or equal to), e.g. 1 <= 1 is true

!= (not equal to), e.g. 8 != 7 is true

<= (less than or equal to), e.g. 6 <= 2 is false

! (not, negation), e.g. !true is false

Complex expressions

Two boolean operators:

- && is the AND operations
- || is the OR operations

p	q	p && q
T	T	T
T	F	F
F	T	F
F	F	F

p	q	p || q
T	T	T
T	F	T
F	T	T
F	F	F

Complex expressions

AND operation removes Ts from the result
The OR operation adds Ts to the result

Evaluate (!p OR q) AND (p)

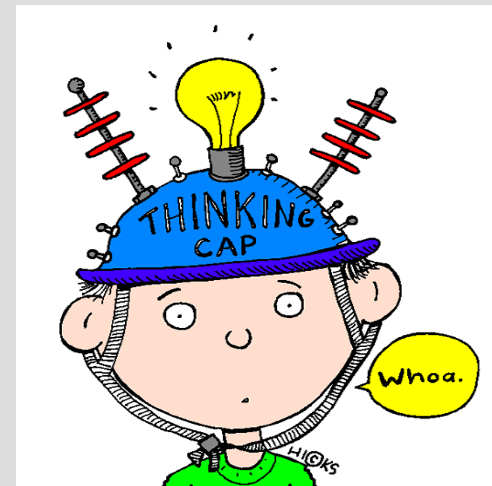
p	q	!p	!p OR q	(!p OR q) AND (p)
T	T	F	T	T
T	F	F	F	F
F	T	T	T	F
F	F	T	T	F

Complex expressions

Write an if statement for checking if a variable (int) x is a positive odd number.

Hint: You may want to use the remainder (also called modulus) operator (the % sign).

For example, 5 % 3 = 2



Complex expressions

Humans tend to use the english word OR to describe XOR (exclusive or)

“We can have our final exam on the scheduled day (May 13) or the last day of class (May 6).”

Did you think the statement above meant final exams on both days was a possibility?

; and if

Please always put {} after if-statements

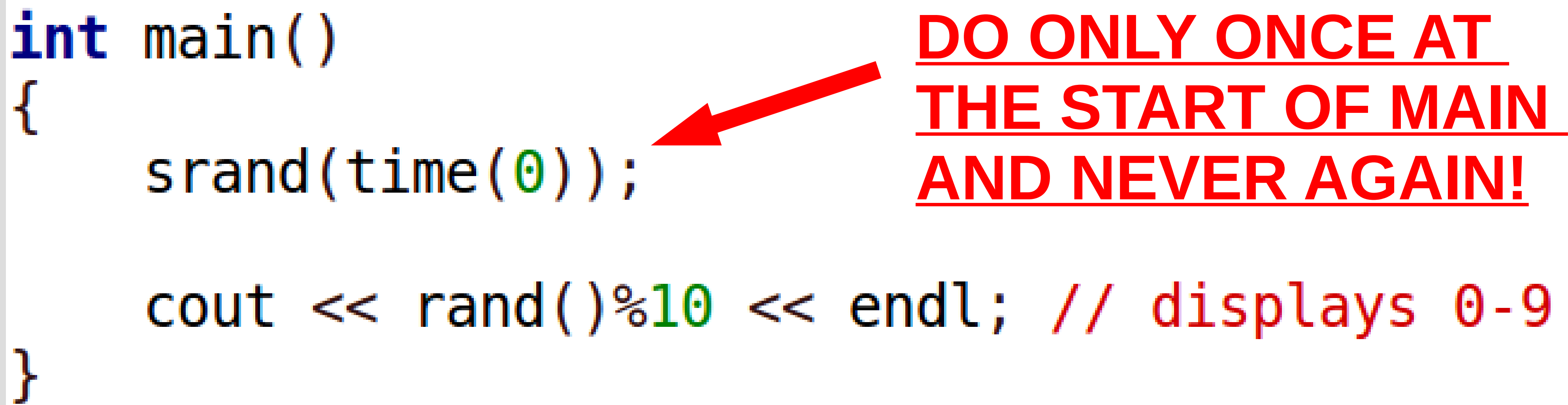
The compiler will let you get away with not putting these (this leads to another issue)

If you do not put {} immediately after an if, it will only associate the first command after with the if-statement
(see: ifAndSemi.cpp)

Random numbers

To use random numbers, you need to do:

1. Run srand(time(0)) once
2. Use rand() to actually generate a number



(See: rng.cpp)

Complex expressions

If statements for when x...

... is between 10 and 20 (inclusive)

```
if(10 <= x && x <= 20)
```

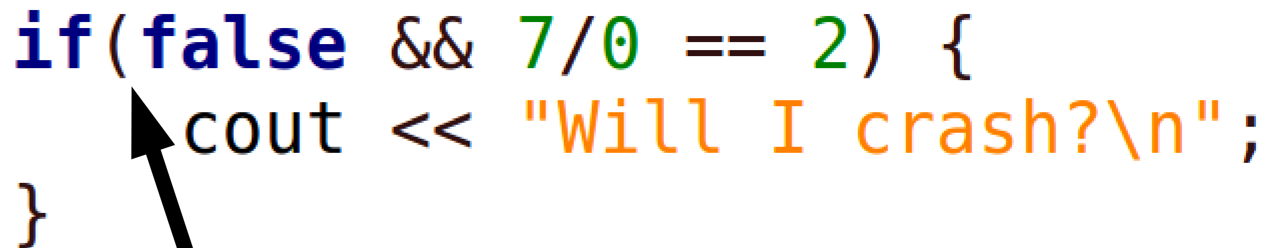
Cannot say: 10 <= x <= 20 (why?)

... is a vowel (x is type char)

```
if( x == 'a' || x == 'e' || x == 'i' || x == 'o' || x == 'u')
```

Short-circuit evaluation

Short-circuit evaluation is when you have a complex bool expression (&& or ||) but you don't need to compute all parts.



```
if(false && 7/0 == 2) {
    cout << "Will I crash?\n";
}
```

If this is false, then it will not check next

(See: shortCircuit.cpp)

Short-circuit evaluation

Simple cases of short-circuit:

When you have a bunch of ORs

```
if( expression || exp || exp || exp )
```

Once it finds any true expression, if statement will be true

When you have a bunch of ANDs

```
if( expression && exp && exp && exp )
```

Once it finds any false expression, if statement will be false

Complex expressions

Be careful when negating, that you follow De Morgan's Law:

bool a, b;
!(a OR b) is equivalent to (!a) AND (!b)
!(a AND b) is equivalent to (!a) OR (!b)

"Neither rainy or sunny" means
"Both not rain and not sunny"

Nested if statements

You can have as many if statements inside each other as you want.

```
if (teacherAwake)
{
    if (studentAwake)
    {
        if (classWellPrepared)
        {
            learning = true;
        }
    }
}
```

Nested if statements

From a truth table perspective, nested loops are similar to AND

The previous if code is equivalent to:

```
if(teacherAwake && studentAwake && classWellPrepared)
{
    learning = true;
}
```

However, sometimes you want to do other code between these evaluations

Nested if statements

(See: bridgeOfDeath.cpp)

Scope

Where a variable is visible is called its <u>scope</u>

Typically variables only live inside the block (denoted with matching { and })

A variable lives until the block is closed, so inner blocks can see everything from the block it was created inside

Scope

```
5   int main()
6   {
7       int x;
8       // can use x here
9       {
10          int y;
11          // can use x or y here
12      }
13      // can use x here
14      return 0;
15  }
```

(See: scope.cpp)

Multiway if/else

This is a special format if you put an
if statement after an else.

This second “if statement” only is tested
when the first “if statement” is not true

(See: grades.cpp)

Switch

A switch statement checks to see if a variable has a specific value.

```
switch( controlingVariable)
{
    case 2:
    case 4:
        cout << "controllingVariable is either 2 or 4" << endl;
        break;
    case 3:
        cout << "controllingVariable is 3\n";
        break;
    default;
        cout << "controllingVariable is not 2, 3 or 4...\n";
        break;
}
```

Controlling Variable

Case label

Break statement

Switch

If the value of the controlling variable
is found in a case label, all code until
a break statement is ran (or the switch ends)

Switch statements only test equality
with case labels (not greater or less than)

(See: switch.cpp)

Switch

Switch statements can be written as multiway if/else statements.

Could use just "if statements" but "else if" shows only one of these will run

(See: switchToIf.cpp)

Conditional operator

We will not use in this class, but if you use other people's code you will encounter

Shorthand for an if-else statement

(boolean) ? [if true] : [if false]

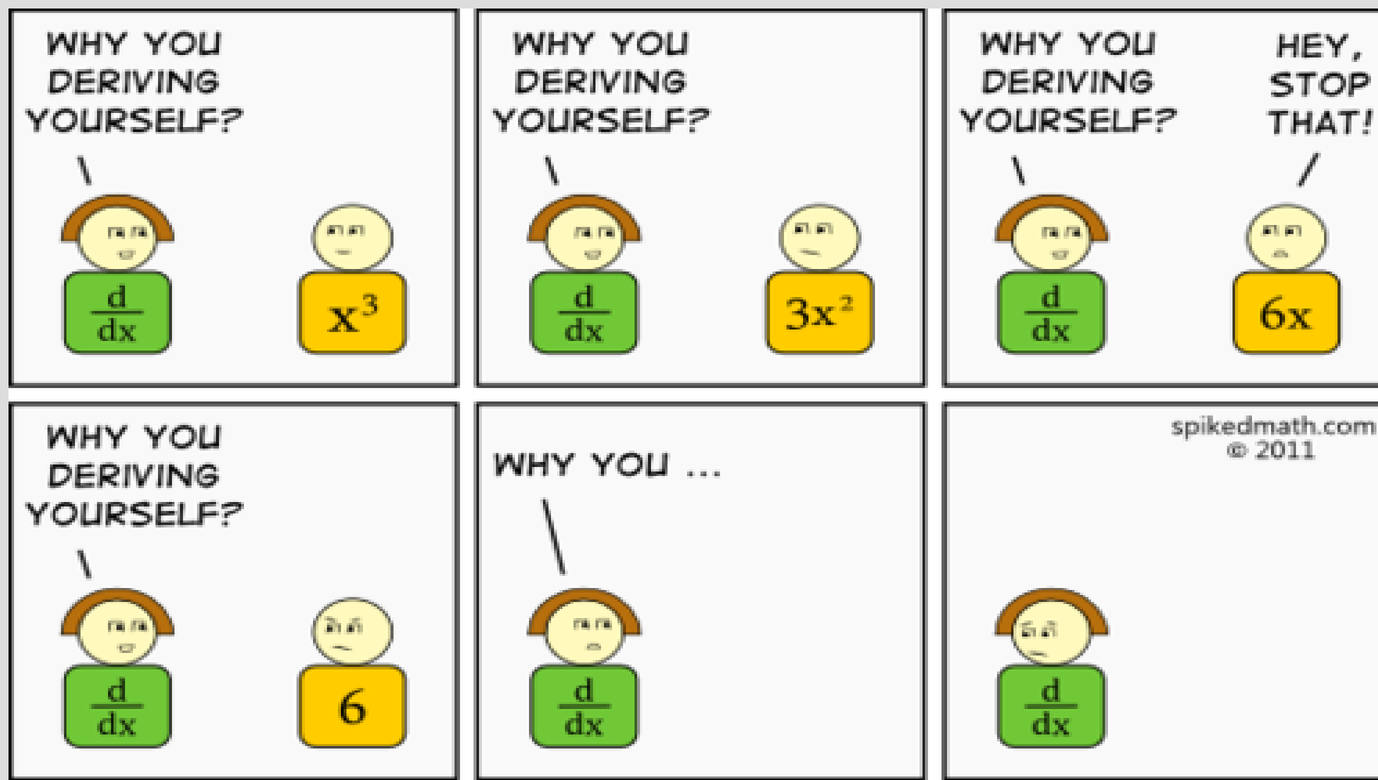
Example:

```
max = (x>y) ? x : y;
```

(See: max.cpp)

Loops

Ch 3.3-3.4



if/else vs loops

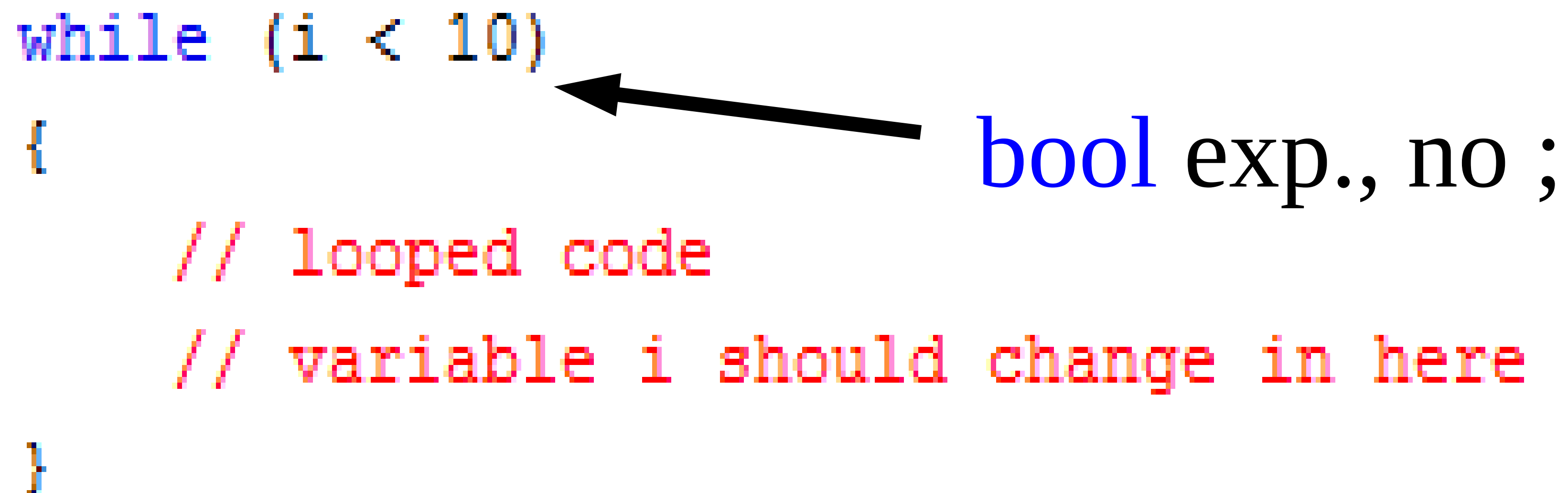
if/else statements makes code inside only sometimes run

Loops make code inside run more than once

Both use boolean expressions to determine if the code inside is run

while loop

A while loop tests a bool expression and will run until that expression is false



(See: whileLoop.cpp)

while loop

The bool expression is tested when first entering the while loop

And!

When the end of the loop code is reached (the } to close the loop)

```
int i = 0;
while (i < 5) {
    cout << "Looping, i = " << i << "\n";
    i++;
}
cout << "Outside the loop, i = " << i << "\n";
```

while loop

It can be helpful to manually work out what loops are doing and how variables change in each loop iteration

This will build an insight into how loops work and will be beneficial when working with more complicated loops

while loop

3 parts to any (good) loop:

- Test variable initialized

```
i=0;
```

- bool expression

```
while (i < 10)
```

- Test variable updated inside loop

```
i++;
```

for loop

A <u>for loop</u> is a compacted version of the while loop (the 3 important parts are together)

for loops are used normally when iterating over a sequence of numbers (i.e. 1, 2, 3, 4)

```
for (int i=0; i < 3; i++)
```

Initialization

boolean expression

Update

(See: forLoop.cpp)

do-while loop

A do-while loop is similar to a normal while loop, **except** the bool expression is only tested at the end of the loop (not at the start)

```
8       cout << "How many times do you want to run the loop?\n";
9       cin >> i; // what happens if i is less than 1?
10      do {
11          cout << "Looping, i = " << i << "\n";
12          i--;
13      } while (i > 0);
14      cout << "Outside the loop, i = " << i << "\n";
```

Note semicolon!

(See: doWhile.cpp)

do-while loop

Q: Why would I ever want a do-while loop?

A: When the first time the variable is set is inside the loop.

You can initialize the variable correctly and use a normal while loop, but this makes the logic harder

Loops

99 bottles of beer on the wall, 99 bottles of beer!
Take one down, pass it around, 98 bottles of beer on the wall!

98 bottles of beer on the wall, 98 bottles of beer!
Take one down, pass it around, 97 bottles of beer on the wall!

97 bottles of beer on the wall, 97 bottles of beer!
Take one down, pass it around, 96 bottles of beer on the wall!
...

Write a program to output the above song
(See 99beer.cpp)

continue

There are two commands that help control loops:

continue tells the loop to start over again

break stops the loop

continue

continue command can be issued to start at the next iteration of a loop

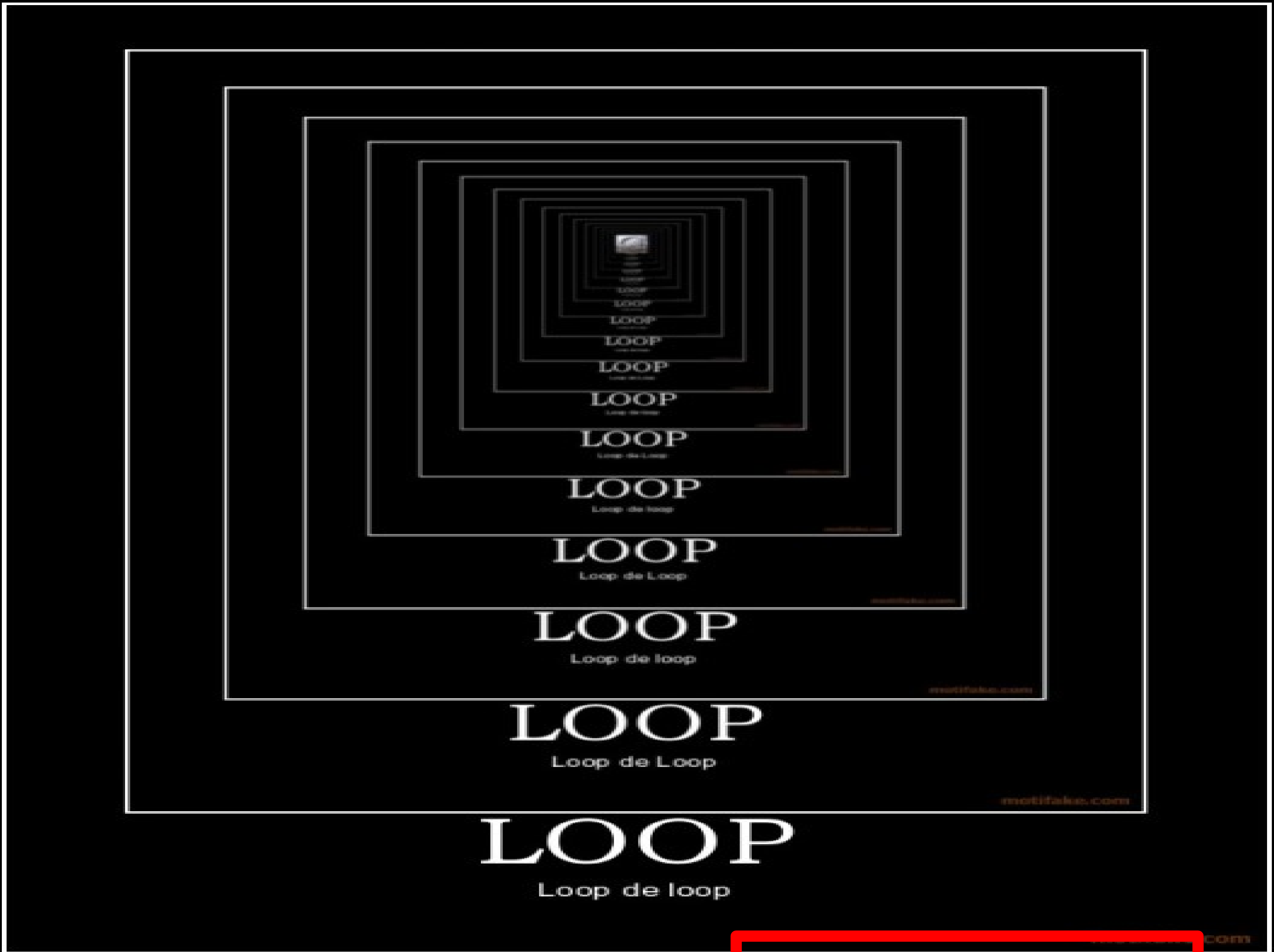
doSkip true

```
for (i = 0; i < 10; i++)
{
    // code will run everytime

    if (doSkip)
    {
        continue;
    }

    // code will not run
    // if doSkip is true
}
```

(See: continue.cpp)



C-C-COMBO BREAKER

motifake.com

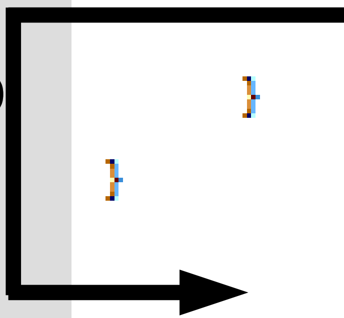
break

break will exit the current loop

```
for (i = 0; i < 10; i++)
{
    // code

    if (doSkip)
    {
        break;
    }
}

// outside loop code
```



doSkip
true

(See: break.cpp)

Infinite loops

(See: countingSheep.cpp)

while loop

https://www.youtube.com/watch?v=7-Nl4JFDLOU

Loops to sum

Loops allow you to decide how many times
a piece of code should run on the fly
(i.e. at run time, not compile time)

You can either directly prompt the user
how many times or make a special value
to “exit” on

(See: sumLoop.cpp)

Debugging

When your program is not working, it is often helpful to add cout commands to find out what is going on

Normally displaying the value of your variables will help you solve the issue

Find up until the point where it works, then show all the values and see what is different than you expected