Week 4

Kyle Dewey

Overview

- New office hour location
- `while` / `do-while` / `for` loops
- `break` / `continue`
- Termination
- Exam recap

Office Hour

Motivation

- Factorial
  - `5! = 5 * 4 * 3 * 2 * 1 = 120`

```
int factorial( int x ) {
   // return x! (x factorial)
}
```

Factorial

- Need a way to say "for each integer from x down to 1, decrementing by 1 at a time"

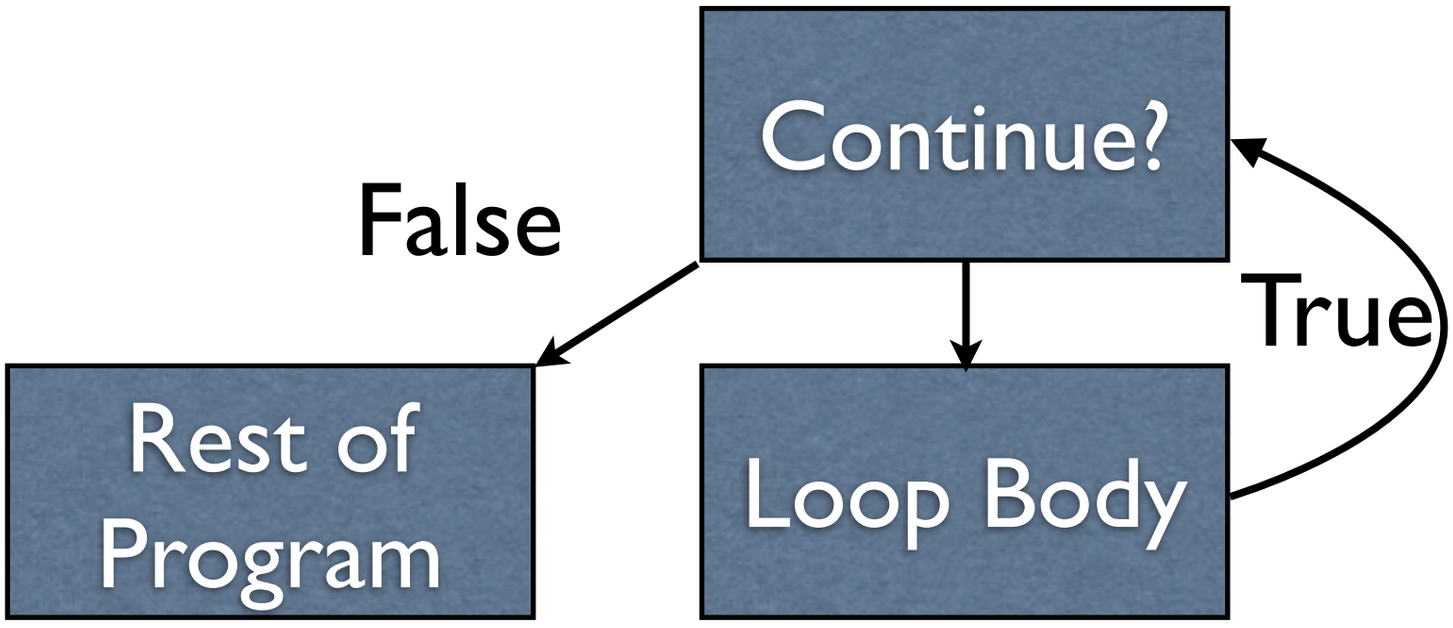
```
5! = 5 * 4 * 3 * 2 * 1 = 120
```

while

```
int x = 0;
while ( x < 5 )
   oneStatement();

while ( x < 5 ) {
   statementOne();
   statementTwo();
}
```

`while` Semantics



Example #1

- What does `x` equal at loop end?

```
int x = 0;

while ( x < 5 ) {
   x++;
}
```

Example #2

- What does `x` equal at loop end?

```
int x = 0;

while ( x < 5 ) {
   x = x + 2;
}
```

Example #3

- What does `x` equal at loop end?
- What does this print?

```
int x = 0;

while ( x < 5 ) {
   x = 10;
   printf( “moo” );
}
```

Factorial Revisited

- A lot of different ways to implement it
- One possible way:
  - Track which number we are looking at in one variable
  - Track the result in another

```
5! = 5 * 4 * 3 * 2 * 1 = 120
```

Factorial Revisited

```
int factorial( int fact ) {
  int result = fact;
  int num = fact - 1;
  while ( num > 1 ) {
    result = result * num;
    num--;
  }

  return result;
}
```

5! = 5 * 4 * 3 * 2 * 1 = 120

Factorial Revisited

```
int factorial( int fact ) {
   int result = 1;
   int num = 2;
   while( num <= fact ) {
      result = result * num;
      num++;
   }

   return result;
}
```

5! = 1 * 2 * 3 * 4 * 5 = 120

do / while

do / while

- Essentially a `while` loop with the condition at the end
- The body will always execute at least once

```
int x = 0;
do {
   statementOne();
   statementTwo();
} while ( x > 0 );
```

Note semicolon!

Example #1

- What does `x` equal at loop end?

```
int x = 0;

do {
   x++;
} while ( x < 5 );
```

Example #2

- What does `x` equal at loop end?

```
int x = 0;

while ( x < 0 ) {
   x++;
}
```

```
int x = 0;

do {
   x++;
} while ( x < 0 );
```

for

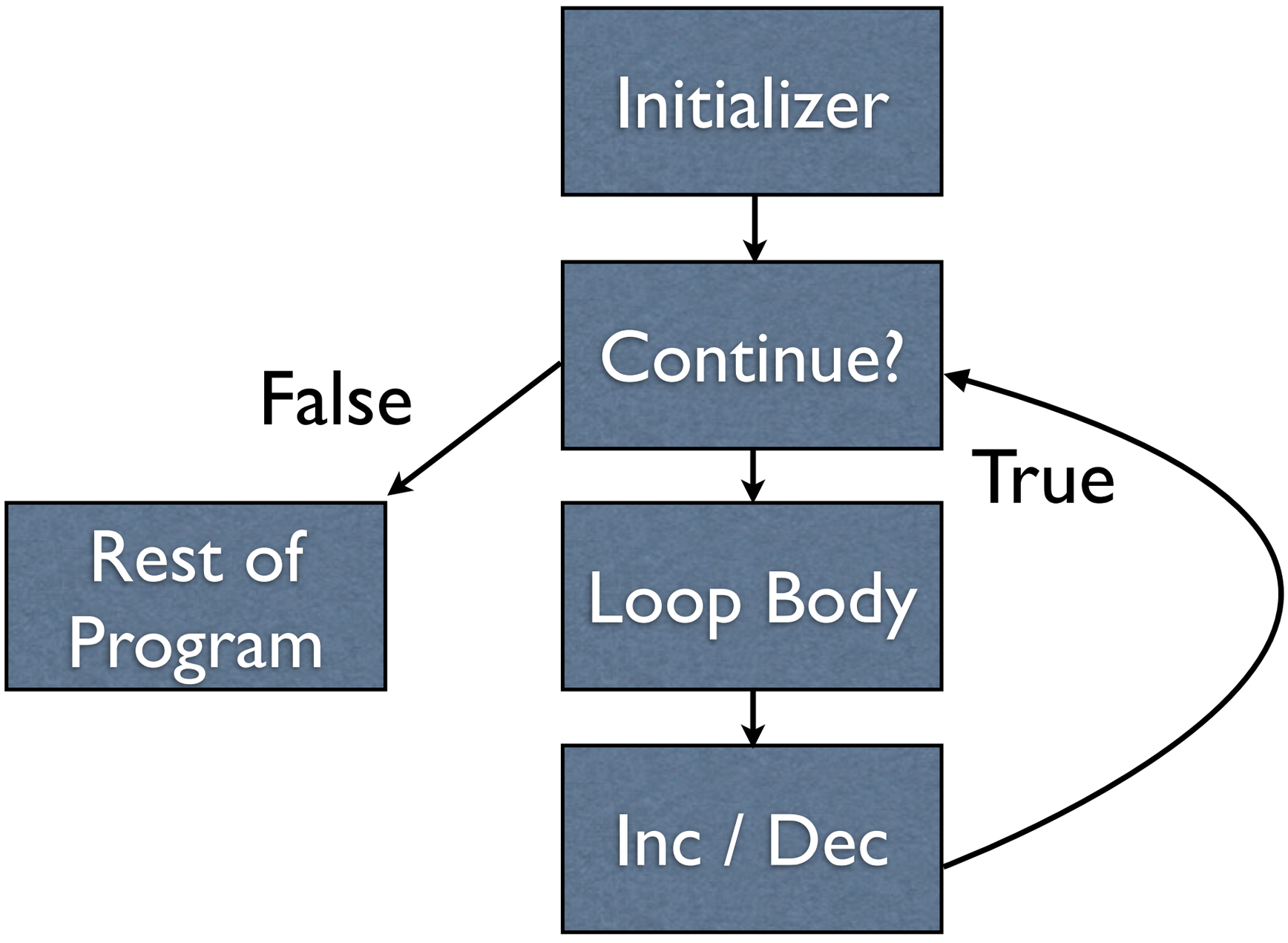
for

```
int x;

for ( x = 0; x < 50; x++ )
  oneStatement( x );

for( x = 0; x < 50; x++ ) {
  statementOne();
  statementTwo();
}
```

`for` Semantics

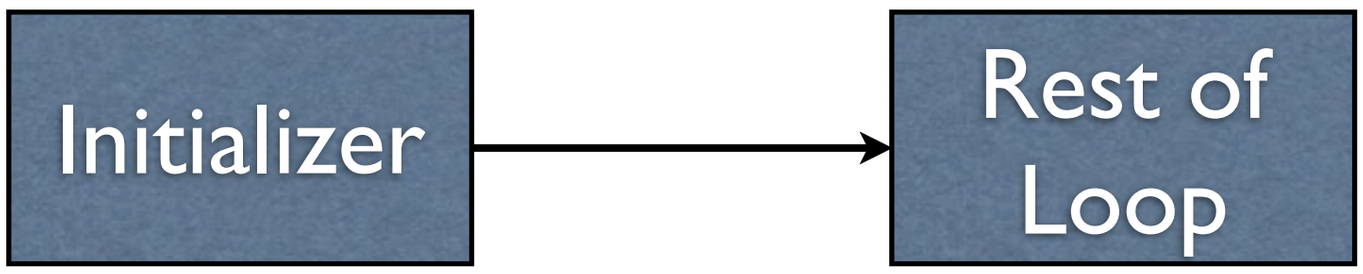


Initializer

```
int x;

for ( x = 0; x < 50; x++ )
    oneStatement( x );
```

- Run before the loop starts

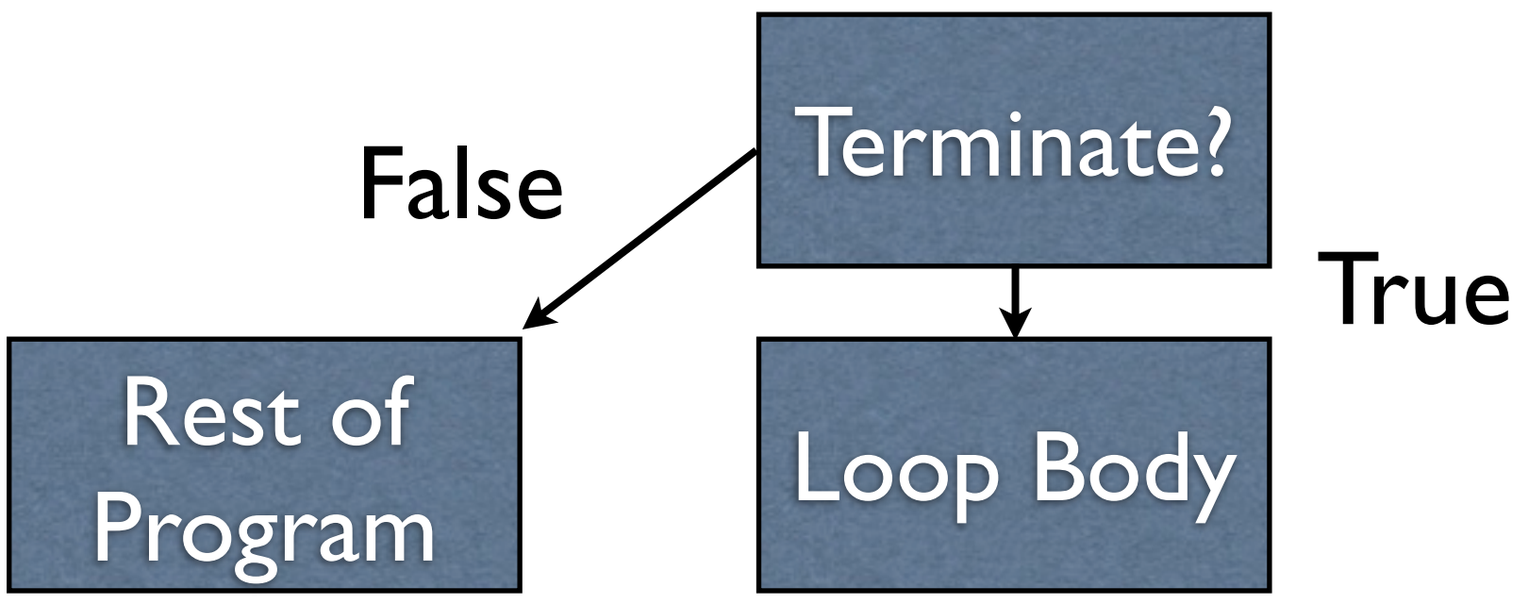


Compare

```
int x;

for ( x = 0; x < 50; x++ )
   oneStatement( x );
```

- Run before each **iteration** of the loop

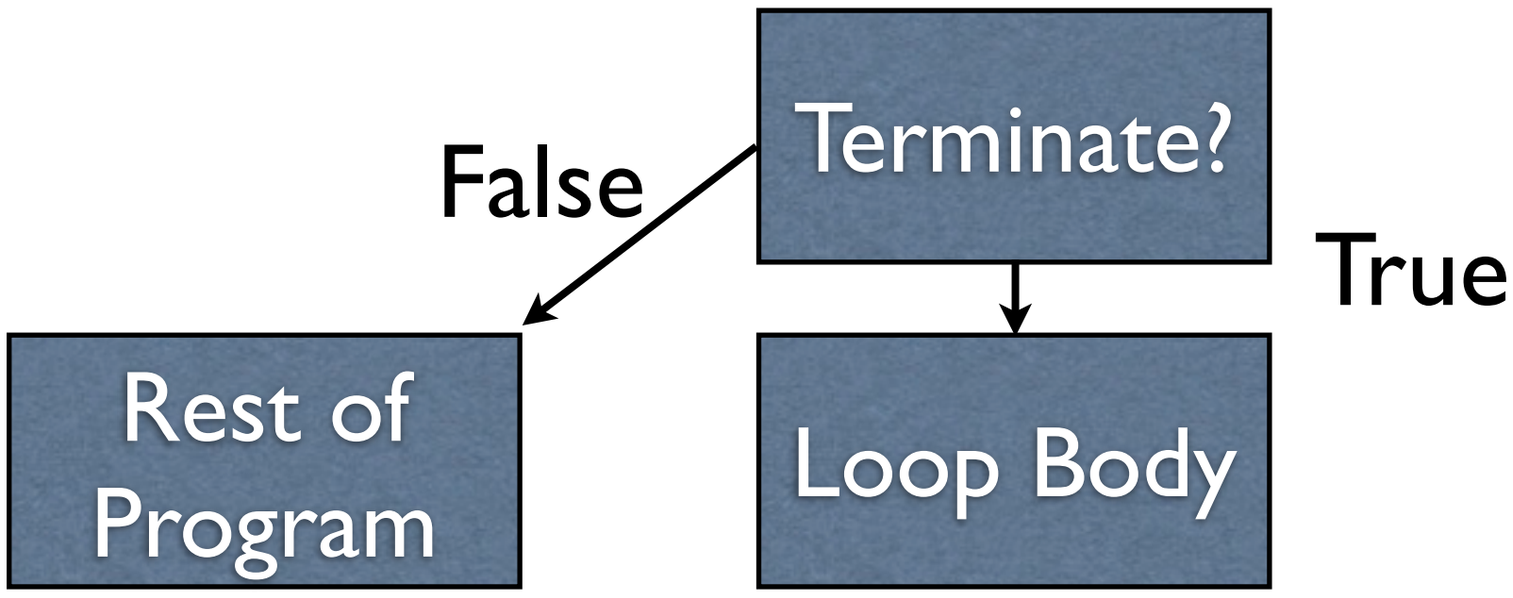


Loop Body

```
int x;

for ( x = 0; x < 50; x++ )
   oneStatement( x );
```

- Run during each **iteration** of the loop

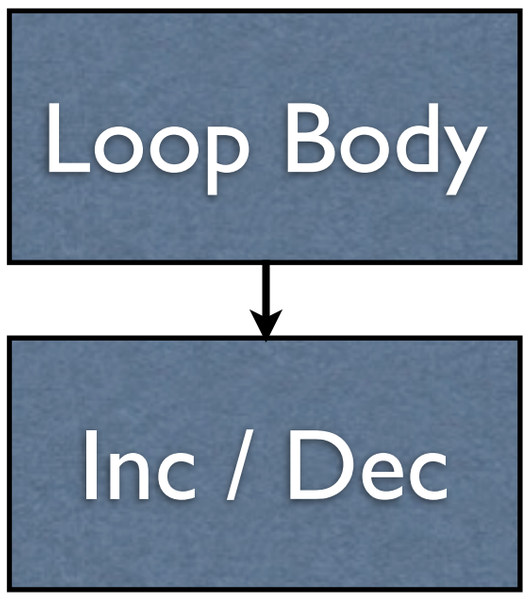


Increment / Decrement

```
int x;

for ( x = 0; x < 50; x++ )
   oneStatement( x );
```

- Run after each **iteration** of the loop



Example #1

- What does this print?

```
int x;

for ( x = 0; x < 5; x++ ) {
  printf( “foobar” );
}
```

Example #2

- What does this print?

```
int x;

for ( x = 0; x < 5; x++ ) {
  printf( "%i\n", x );
}
```

Example #3

- What does `y` equal at loop end?

```
int x;
int y = 0;

for ( x = 0; x < 5; x++ ) {
  y++;
}
```

Example #4

- What does `y` equal at loop end?

```
int x;
int y = 0;

for ( x = 1; x < 4; x++ ) {
  y++;
}
```

Example #5

- What does `y` equal at loop end?

```
int x;
int y = 0;

for ( x = 1; x % 3 != 0; x++ ) {
   y++;
}
```

`for` Header

- It is not required to specify an increment, compare, or counter
- The semicolon still needs to be provided
- Can be tricker to read and understand

for Header

```
int x;

for ( x = 0; x < 5; x++ ) {
  printf( “moo” );
}
```

- ...is effectively the same as...

```
int x = 0;

for ( ; x < 5; x++ ) {
  printf( “moo” );
}
```

`for` Header

```
int x;

for ( x = 0; x < 5; x++ ) {
  printf( "moo" );
}
```

- ...is effectively the same as...

```
int x = 0;

for ( ; x < 5; ) {
  printf( "moo" );
  x++;
}
```

Factorial Revisited

- A lot of different ways to implement it
- One possible way:
  - Track which number we are looking at in one variable
  - Track the result in another

```
5! = 5 * 4 * 3 * 2 * 1 = 120
```

Factorial Revisited

```
int factorial( int fact ) {
   int result = fact;
   int num;
   for( num = fact - 1; num > 1; num-- ){
      result = result * num;
   }

   return result;
}
```

5! = 5 * 4 * 3 * 2 * 1 = 120

Factorial Revisited

```
int factorial( int fact ) {
   int result = 1;
   int num;
   for( num = 2; num <= fact; num++ ){
      result = result * num;
   }

   return result;
}
```

5! = 1 * 2 * 3 * 4 * 5 = 120

`break`

- `break` can be used to immediately exit from a loop
- Can make things easier when used carefully

Example #1

```
int x = 0;
while ( x < 5 ) {
   break;
}
```

Example #2

```
int x = 0;
while ( x < 5 ) {
    x = x + 2;
    if ( x >= 5 )
      break;
    printf( “moo” );
}
```

Example #3

```
int x;
for ( x = 0; x < 5; x = x + 2 ) {
   if ( x >= 5 )
     break;
   printf( “moo” );
}
```

`continue`

- `continue` causes the loop to skip the rest of the body
- Increments / decrements still performed for `for` loops
- Conditions will be rechecked

Example #1

```
int x = 0;
while ( x < 10 ) {
   x++;
   continue;
   printf( “moo” );
}
```

Example #2

```
int x;
for( x = 0; x < 10; x++ ) {
   if ( x % 2 == 0 )
      continue;
   printf( "%i\n", x );
}
```

Termination

- Consider the following:

```
while ( 1 ) {
  printf( “moo\n” );
}
```

Termination

- Some loops may never terminate
- This may be desired
  - ...or not

Termination

- Useful example

```
char* command;

while ( 1 ) {
   printCommandPrompt();
   command = readCommand();
   executeCommand( command );
}
```

Termination

- Likely an error
- ...or perhaps a clever way of checking for overflow and maximum `int` size

```
int x = 0;
int y = 1;

while ( x < y ) {
   x++;
   y++;
}
```

Infinite Loops

- Loops that never terminate are called infinite loops
- Usually in this context it is a bug
- Can be nasty to debug
  - Complex termination conditions
  - Is it just taking awhile?

Random Notes on Loops

Intentional Nontermination

- `while` loop:

```
while ( 1 ) { ... }
```

- `for` loop:

```
for ( ;; ) { ... }
```

`for` **vs.** `while`

- Each can be expressed in terms of the other
- Choose based on what seems more appropriate

while **as** for

```
while ( x < 5 ) { ... }
```

- Can be represented as:

```
for (; x < 5;) { ... }
```

for **as** while

```
int x;
for ( x = 0; x < 5; x++ ) { ... }
```

- Can be represented as: (except for `continue` **behavior!)**

```
int x = 0;
while ( x < 5 ) {
   ...;
   x++;
}
```

Exam Recap

Difficulty Level

- Too hard?
- Too easy?
- Just right?

Grading

- Hopefully by Tuesday, if not Tuesday then Thursday
- Depending on how the grades turn out:
  - No curve
  - Curve
  - Change weights
- Your grade can only improve with said changes

Exam Solutions