CS12 CH:9 For Loops

Iterative Control Structures

Mr. Gullo

October 27, 2025

Learning Objectives

By the end of this lesson, students will be able to:

- Construct for loops with proper syntax (initialization, condition, update)
- Explain when to use for loops versus while loops
- Understand variable scope within loop structures
- Apply compound assignment operators (+=, -=, *=, /=)
- Debug common for loop syntax errors
- Implement nested loops for multi-dimensional iterations
- Convert between while and for loop structures

For Loops: Why Have Them?

For loops are designed for counting through known values.

For Loops: Why Have Them?

For loops are designed for counting through known values.

Best use cases:

- Repeat something a specific number of times (e.g., 10 times)
- Move from first element to last element of a list
- Move through each letter in a word or sentence

For Loops: Why Have Them?

For loops are designed for counting through known values.

Best use cases:

- Repeat something a specific number of times (e.g., 10 times)
- Move from first element to last element of a list
- Move through each letter in a word or sentence

While loops are better for:

- Unknown number of repetitions
- Getting user input
- Problems involving division or multiplication
- Waiting for a condition change

Loop Flow Diagram Context

Both while and for loops follow the same fundamental control flow pattern.

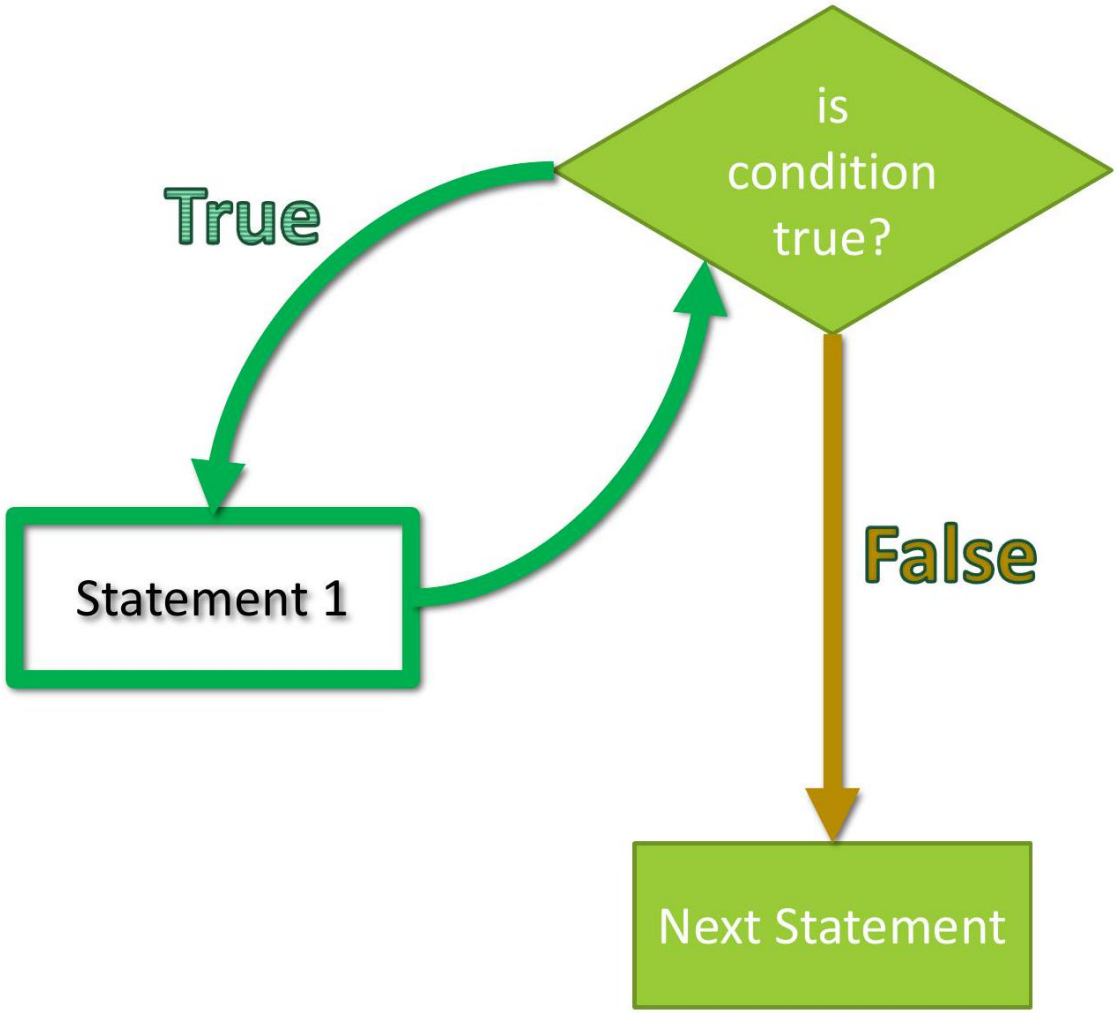
Loop Flow Diagram Context

Both while and for loops follow the same fundamental control flow pattern. The key difference is syntax organization: for loops package initialization, condition, and update in one line.

Loop Flow Diagram Context

Both while and for loops follow the same fundamental control flow pattern.
The key difference is syntax organization: for loops package initialization, condition, and update in one line.
Next slide shows the visual flow that applies to both loop types.

Loop Flow Diagram



This control flow applies equally to while and for loops.

For Loop Syntax

Example: Count to 10

```
#include <iostream>
using namespace std;

int main() {
    for(int i = 1; i <= 10; i++)
        cout << i << endl;

    return 0;
}
```

For Loop Syntax

Example: Count to 10

```
#include <iostream>
using namespace std;

int main() {
    for(int i = 1; i <= 10; i++)
        cout << i << endl;

    return 0;
}
```

Output: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 (each on separate line)

For Loop Components Breakdown

```
for(int i = 1; i <= 10; i++)
```

For Loop Components Breakdown

```
for(int i = 1; i <= 10; i++)
```

Three components (separated by semicolons):

1. `int i = 1;` - **Initialization:** Sets counter starting value

For Loop Components Breakdown

```
for(int i = 1; i <= 10; i++)
```

Three components (separated by semicolons):

1. `int i = 1;` - **Initialization:** Sets counter starting value
2. `i` $\leq$ `10;` - **Condition:** Loop continues while true

For Loop Components Breakdown

```
for(int i = 1; i <= 10; i++)
```

Three components (separated by semicolons):

1. `int i = 1;` - **Initialization:** Sets counter starting value
2. `i` $\leq$ `10;` - **Condition:** Loop continues while true
3. `i++` - **Update:** Runs at end of each iteration

Variable Scope: Context

Variables declared in the for loop header have limited lifetime.

Variable Scope: Context

Variables declared in the for loop header have limited lifetime.
They only exist within the loop structure itself.

Variable Scope: Context

Variables declared in the for loop header have limited lifetime.
They only exist within the loop structure itself.
Next slide illustrates what happens when you try to access a loop variable outside its scope.

Variable Scope: The Loop Variable

```
for(int i = 1; i <= 10; i++)
    cout << i << endl;

cout << i << endl;  // ERROR!
```

Variable Scope: The Loop Variable

```
for(int i = 1; i <= 10; i++)
    cout << i << endl;

cout << i << endl;  // ERROR!
```

Compiler error:

```
error: 'i' was not declared in this scope
```

Variable Scope: The Loop Variable

```
for(int i = 1; i <= 10; i++)
    cout << i << endl;

cout << i << endl;  // ERROR!
```

Compiler error:

```
error: 'i' was not declared in this scope
```

The variable `i` only exists inside the for loop. It is destroyed after the loop ends.

Scope Visualization Context

Understanding scope helps prevent common programming errors.

Scope Visualization Context

Understanding scope helps prevent common programming errors. Think of scope as a boundary: variables created inside cannot escape outside.

Scope Visualization Context

Understanding scope helps prevent common programming errors.
Think of scope as a boundary: variables created inside cannot escape outside.
Next slide shows a visual representation of this concept.

Scope Visualization

main() scope

for loop scope

```
int i
```

Converting While to For: Leap Years

While loop version:

```
int y1 = 1893;
int y2 = 1915;
while(y1 <= y2) {
    if(y1 % 400 == 0 || (y1 % 4 == 0 && y1 % 100 != 0))
        cout << y1 << endl;
    y1++;
}
```

Converting While to For: Leap Years

While loop version:

```
int y1 = 1893;
int y2 = 1915;
while(y1 <= y2) {
    if(y1 % 400 == 0 || (y1 % 4 == 0 && y1 % 100 != 0))
        cout << y1 << endl;
    y1++;
}
```

For loop version:

```
int y1 = 1893;
int y2 = 1915;
for(int i = y1; i <= y2; i++) {
    if(i % 400 == 0 || (i % 4 == 0 && i % 100 != 0))
        cout << i << endl;
}
```

Converting While to For: Arithmetic Sequence

For loop version:

```
int t = 12;
int d = -2;
int n = 10;
int sum = t;

cout << "Term\t\tSum\n";

for(int i = 1; i <= n; i++) {
    cout << t << "\t\t" << sum << endl;
    t = t + d;
    sum = sum + t;
}
```

More concise than while loop when you know the exact iteration count.

Compound Assignment Operators

C++ provides shortcuts for common operations.

Compound Assignment Operators

C++ provides shortcuts for common operations.

Operator	Example	Equivalent to
+=	`x += 6;`	`x = x + 6;`
-=	`x -= 10;`	`x = x - 10;`
*=	`x *= 2;`	`x = x * 2;`
/=	`x /= 2;`	`x = x / 2;`

Compound Assignment Operators

C++ provides shortcuts for common operations.

Operator	Example	Equivalent to
+=	`x += 6;`	`x = x + 6;`
-=	`x -= 10;`	`x = x - 10;`
*=	`x *= 2;`	`x = x * 2;`
/=	`x /= 2;`	`x = x / 2;`

These perform the operation AND store the result in one step.

Compound Operators: Examples

```
int x = 10;

x += 6;    // x is now 16
x -= 10;   // x is now 6
x *= 2;    // x is now 12
x /= 2;    // x is now 6
```

Compound Operators: Examples

```
int x = 10;

x += 6;    // x is now 16
x -= 10;   // x is now 6
x *= 2;    // x is now 12
x /= 2;    // x is now 6
```

Common usage in loops:

```
sum += value;  // Add to running total
count++;       // Increment counter
```

Common Syntax Error: Commas

WRONG (commas):

```
for(int i = 0, i < 10, i++) {
    cout << i << '\n';
}
```

Common Syntax Error: Commas

WRONG (commas):

```
for(int i = 0, i < 10, i++) {
    cout << i << '\n';
}
```

Compiler error: Will not compile!

Common Syntax Error: Commas

WRONG (commas):

```
for(int i = 0, i < 10, i++) {
    cout << i << '\n';
}
```

Compiler error: Will not compile!

CORRECT (semicolons):

```
for(int i = 0; i < 10; i++) {
    cout << i << '\n';
}
```

Remember: semicolons separate the three for loop components.

Nested Loops: Context

Loops can be placed inside other loops, creating nested structures.

Nested Loops: Context

Loops can be placed inside other loops, creating nested structures.
Each iteration of the outer loop runs the entire inner loop.

Nested Loops: Context

Loops can be placed inside other loops, creating nested structures.
Each iteration of the outer loop runs the entire inner loop.
This is useful for working with multi-dimensional data like grids, tables, or coordinate systems.

Nested Loops: Context

Loops can be placed inside other loops, creating nested structures.
Each iteration of the outer loop runs the entire inner loop.
This is useful for working with multi-dimensional data like grids, tables, or coordinate systems.
Next slides show examples with different nesting depths.

Nested Loops: Double Loop Example

Predict the output:

```
int base = 8;
for(int i = 0; i < base; i++) {
    for(int j = 0; j < base; j++) {
        cout << i << "," << j << endl;
    }
}
```

Nested Loops: Double Loop Example

Predict the output:

```
int base = 8;
for(int i = 0; i < base; i++) {
    for(int j = 0; j < base; j++) {
        cout << i << "," << j << endl;
    }
}
```

Explanation:

- Outer loop (i) runs 8 times
- For each i, inner loop (j) runs 8 times
- Total: $8 \times 8 = 64$ lines of output
- Pattern: (0,0), (0,1), ..., (0,7), (1,0), (1,1), ..., (7,7)

Nested Loops: Triple Loop Example

Predict the output:

```
int testValue = 0;
for(int i = 0; i < 8; i++) {
    for(int j = 0; j < 9; j++) {
        for(int k = 0; k < 10; k++) {
            testValue++;
        }
    }
}
cout << testValue << endl;
```

Nested Loops: Triple Loop Example

Predict the output:

```
int testValue = 0;
for(int i = 0; i < 8; i++) {
    for(int j = 0; j < 9; j++) {
        for(int k = 0; k < 10; k++) {
            testValue++;
        }
    }
}
cout << testValue << endl;
```

Answer: 720

Nested Loops: Triple Loop Example

Predict the output:

```
int testValue = 0;
for(int i = 0; i < 8; i++) {
    for(int j = 0; j < 9; j++) {
        for(int k = 0; k < 10; k++) {
            testValue++;
        }
    }
}
cout << testValue << endl;
```

Answer: 720

Calculation: $8 \times 9 \times 10 = 720$ iterations

Nested Loop Visualization Context

Nested loops create a grid-like iteration pattern.

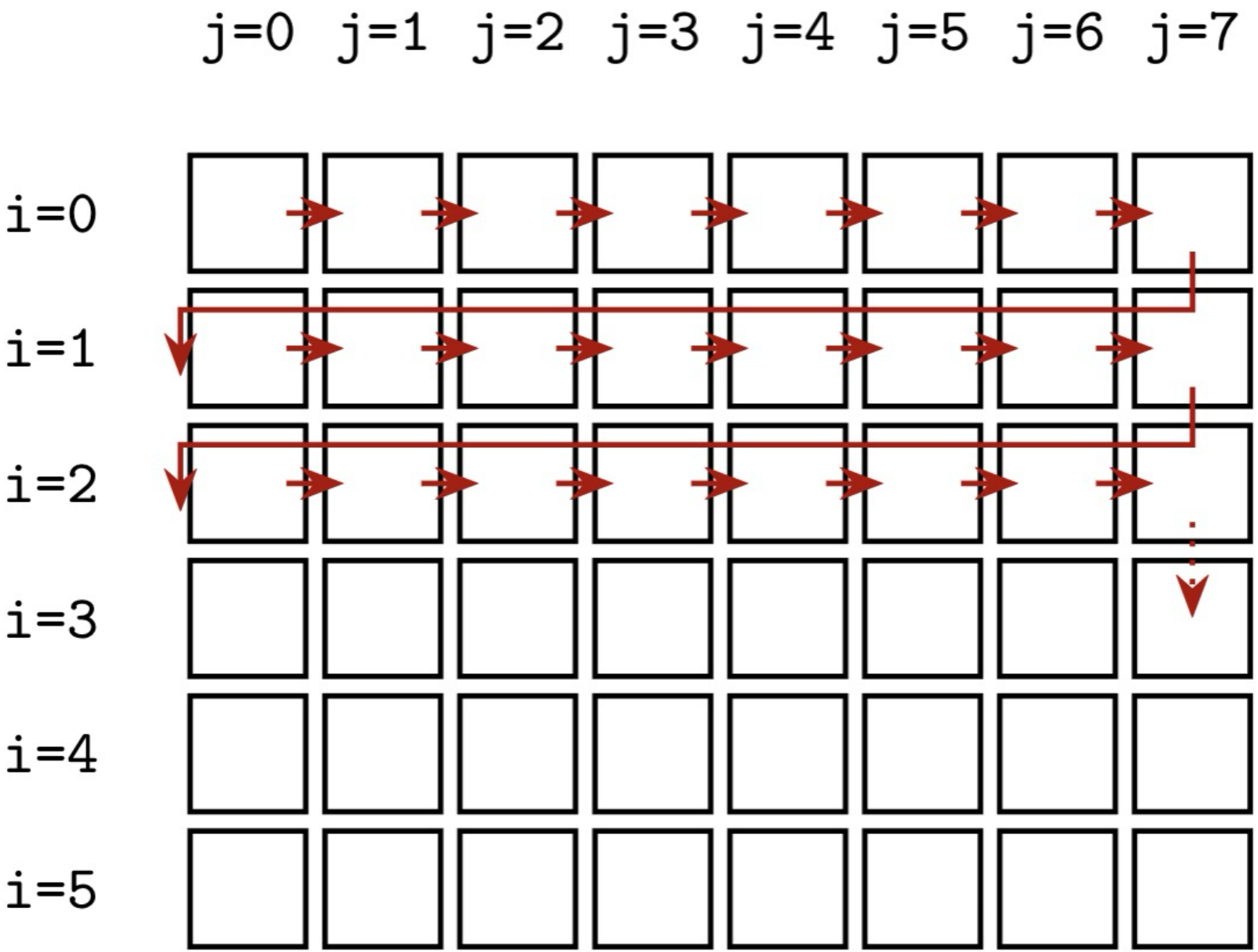
Nested Loop Visualization Context

Nested loops create a grid-like iteration pattern.
The outer loop controls rows, inner loop controls columns.

Nested Loop Visualization Context

Nested loops create a grid-like iteration pattern.
The outer loop controls rows, inner loop controls columns.
Next slide visualizes this concept.

Nested Loop Visualization



Exercise 1: Factors

Exercise File: `01_factors.cpp` (Template with TODOs)
Objective: Output all factors of a positive integer without trailing comma.

Exercise 1: Factors

Exercise File: `01_factors.cpp` (Template with TODOs)
Objective: Output all factors of a positive integer without trailing comma.

```cpp
#include <iostream>
using namespace std;

int main() {
    int n = 24;

    // TODO 1: Create a for loop from 1 to n
    // TODO 2: Check if i divides n evenly (use modulo)
    // TODO 3: Print i followed by comma (except for last)
    // Hint: Check if i equals n to avoid trailing comma

    return 0;
}
```

Expected output for n=24: 1, 2, 3, 4, 6, 8, 12, 24

Exercise 2: Output a Square

Exercise File: `02_square.cpp` (Template with TODOs)
Objective: Use nested loops to create a square of X characters.

Exercise 2: Output a Square

Exercise File: `02_square.cpp` (Template with TODOs)
Objective: Use nested loops to create a square of X characters.

```cpp
#include <iostream>
using namespace std;

int main() {
    int length = 4;

    // TODO 1: Create outer loop for rows (i from 0 to length)
    // TODO 2: Create inner loop for columns (j from 0 to length)
    // TODO 3: Print 'X' in inner loop
    // TODO 4: Print newline after inner loop completes

    return 0;
}
```

Expected output for length=4:

```
XXXX
XXXX
XXXX
XXXX
```

Exercise 3: Output a Triangle

Exercise File: `03_triangle.cpp` (Template with TODOs)
Objective: Create isosceles right triangle using nested loops.

Exercise 3: Output a Triangle

Exercise File: `03_triangle.cpp` (Template with TODOs)
Objective: Create isosceles right triangle using nested loops.

```cpp
#include <iostream>
using namespace std;

int main() {
    int leg = 4;

    // TODO 1: Outer loop for rows (i from leg down to 1)
    // TODO 2: Inner loop for columns (j from 0 to i)
    // TODO 3: Print 'X' in inner loop
    // TODO 4: Print newline after inner loop

    return 0;
}
```

Expected output for leg=4:

```
XXXX
XXX
XX
X
```

Exercise 4: Temperature Conversion

Exercise File: `04_temperature.cpp` (Template with TODOs)
Objective: Display table of 20 Fahrenheit to Celsius conversions.

Exercise 4: Temperature Conversion

Exercise File: `04_temperature.cpp` (Template with TODOs)
Objective: Display table of 20 Fahrenheit to Celsius conversions.

```cpp
#include <iostream>
using namespace std;

int main() {
    float fahrenheit = 20.0;
    float celsius;

    cout << "Fahrenheit\tCelsius\n";

    // TODO 1: Create for loop (i from 0 to 19)
    // TODO 2: Calculate celsius = (5.0/9.0)*(fahrenheit-32.0)
    // TODO 3: Print fahrenheit and celsius
    // TODO 4: Increment fahrenheit by 4

    return 0;
}
```

Start at 20°F, increment by 4°. Use float for decimal precision.

Exercise 5: Sum of Multiples (Project Euler #1)

Exercise File: `05_multiples.cpp` (Template with TODOs)
Problem: Find sum of all multiples of 3 or 5 below 1000.

Exercise 5: Sum of Multiples (Project Euler #1)

Exercise File: `05_multiples.cpp` (Template with TODOs)
Problem: Find sum of all multiples of 3 or 5 below 1000.

```cpp
#include <iostream>
using namespace std;

int main() {
    int sum = 0;

    // TODO 1: Create for loop (i from 1 to 999)
    // TODO 2: Check if i is divisible by 3 OR 5
    // TODO 3: If true, add i to sum
    // TODO 4: Print final sum

    return 0;
}
```

Hint: Number 15 is divisible by both 3 and 5, count it once.

Exercise 6: Even Fibonacci Sum (Project Euler #2)

Exercise File: `06_fibonacci.cpp` (Template with TODOs)
Problem: Sum of even Fibonacci terms below 1,000,000.

Exercise 6: Even Fibonacci Sum (Project Euler #2)

Exercise File: `06_fibonacci.cpp` (Template with TODOs)
Problem: Sum of even Fibonacci terms below 1,000,000.

```cpp
#include <iostream>
using namespace std;

int main() {
    int a = 1, b = 2;
    int sum = 0;

    // TODO 1: While loop: continue while b < 1000000
    // TODO 2: If b is even, add to sum
    // TODO 3: Calculate next Fibonacci: temp = a + b
    // TODO 4: Update a = b, b = temp
    // TODO 5: Print final sum

    return 0;
}
```

Sequence: 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

Exercise 7: Prime Factors (Project Euler #3)

Exercise File: `07_primeFactors.cpp` (Template with TODOs)
Problem: Find largest prime factor of 600851475143.

Exercise 7: Prime Factors (Project Euler #3)

Exercise File: `07_primeFactors.cpp` (Template with TODOs)
Problem: Find largest prime factor of 600851475143.

```cpp
#include <iostream>
using namespace std;

int main() {
    unsigned long long testNum = 600851475143;
    unsigned long long current_n = 2;
    unsigned long long largestPrime = 0;

    // TODO 1: While loop: continue while testNum > 1
    // TODO 2: If testNum divisible by current_n
    // TODO 3:   Update largestPrime = current_n
    // TODO 4:   Divide testNum by current_n
    // TODO 5: Else increment current_n
    // TODO 6: Print largestPrime

    return 0;
}
```

Prime Factors Algorithm Context

Finding prime factors uses division to break down numbers.

Prime Factors Algorithm Context

Finding prime factors uses division to break down numbers.
Algorithm repeatedly divides by smallest possible divisor.

Prime Factors Algorithm Context

Finding prime factors uses division to break down numbers.
Algorithm repeatedly divides by smallest possible divisor.
Because we test divisors in order (2, 3, 4, ...), any divisor that works must be prime. Why?

Prime Factors Algorithm Context

Finding prime factors uses division to break down numbers.
Algorithm repeatedly divides by smallest possible divisor.
Because we test divisors in order (2, 3, 4, ...), any divisor that works must be prime. Why?
If it had smaller factors, we would have already divided them out.

Prime Factors Algorithm Context

Finding prime factors uses division to break down numbers.
Algorithm repeatedly divides by smallest possible divisor.
Because we test divisors in order (2, 3, 4, ...), any divisor that works must be prime. Why?
If it had smaller factors, we would have already divided them out.
Next slide shows conceptual breakdown with smaller number.

Prime Factors: How It Works

Example with 12:

```
testNum = 12, current_n starts at 2

12 ÷ 2 = 6 (remainder 0) → Found prime factor 2
testNum becomes 6

6 ÷ 2 = 3 (remainder 0) → Found prime factor 2
testNum becomes 3

3 ÷ 2 = 1 remainder 1 → Not divisible, try next
current_n becomes 3

3 ÷ 3 = 1 (remainder 0) → Found prime factor 3
testNum becomes 1, DONE
```

Prime factors of 12: 2, 2, 3 (multiply: $2 \times 2 \times 3 = 12$)

Summary

Key takeaways:

- For loops package initialization, condition, update in one line
- Best for known iteration counts
- Loop variables have limited scope (exist only in loop)
- Compound operators (+=, -=, *=, /=) save typing
- Use semicolons, not commas, in for loop header
- Nested loops multiply iteration counts
- Convert while to for when iteration count is known

Summary

Key takeaways:

- For loops package initialization, condition, update in one line
- Best for known iteration counts
- Loop variables have limited scope (exist only in loop)
- Compound operators (+=, -=, *=, /=) save typing
- Use semicolons, not commas, in for loop header
- Nested loops multiply iteration counts
- Convert while to for when iteration count is known

Practice makes perfect: Complete all seven exercises to master for loops!