UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical &
Computer Engineering

ECE 150 *Fundamentals of Programming*

For loops

Prof. Hiren Patel, Ph.D.
Douglas Wilhelm Harder, M.Math. LEL
hdpatel@uwaterloo.ca dwharder@uwaterloo.ca

Outline

- In this lesson, we will:
  - Introduce the concept of repetition and the for loop
  - Look examples using the loop variable
  - Author a program to determine if an integer is prime
  - Consider the different variations of a for loop
  - Look at three examples of a loop within a loop

Repetition statements

- Suppose we wanted to repeat an action a fixed number of times

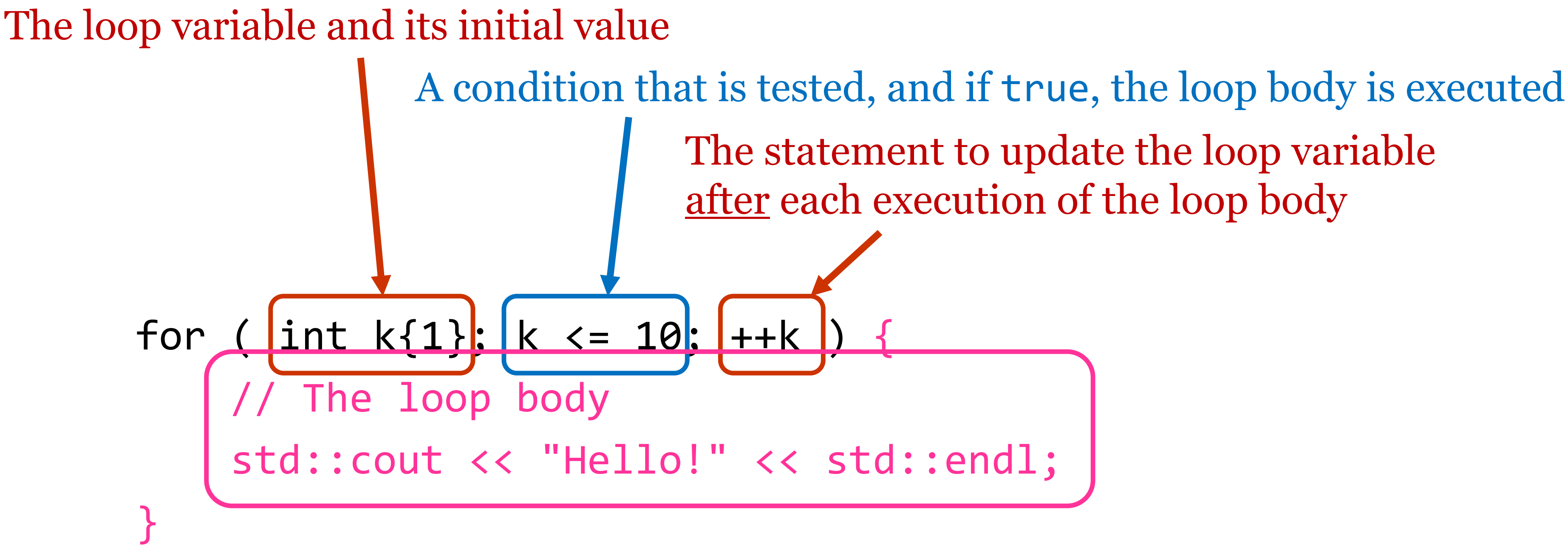
```
#include <iostream>

// Function declarations
int main();

// Function definitions
int main() {
    for ( int k{1}; k <= 10; ++k ) {
        std::cout << "Hello!" << std::endl;
    }

    return 0;
}
```

Output:

```
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
Hello!
```

The components of a for loop

- Looking at the for loop:



The loop variable and its initial value

A condition that is tested, and if `true`, the loop body is executed

The statement to update the loop variable
after each execution of the loop body

```
for ( int k{1}; k <= 10; ++k ) {
    // The loop body
    std::cout << "Hello!" << std::endl;
}
```

Performing a loop

- Working through this example:

```
for ( int k{1}; k <= 5; ++k ) {
    // The loop body
    std::cout << "Hello!" << std::endl;
}
```

  - A loop variable k is initialized with the value

k [illegible]

Output:

```
Hello!
Hello!
Hello!
Hello!
Hello!
```

Calculating *n*!

- Here we calculate the value of 5!

```
int factorial{1};

for ( int k{1}; k <= 5; ++k ) {
    // The loop body
    factorial *= k;
}

std::cout << factorial << std::endl;
```

k	5
factorial	120

Output:

```
120
```

Is *n* prime?

- Let us determine if an integer *n* is prime
  - By definition, *n* is prime if it is divisible only by 1 and *n*
  - In other words, *n* is prime if it is not divisible by 2, 3, …, *n* – 1
  - If *n* is divisible by *k*,

    the remainder of *n* ÷ *k* of zero
  - In C++, we find the remainder of *n* ÷ *k* by calculating `n%k`
  - Therefore, test if `n%k == 0` for `k` going from `2` to `n - 1`

Is *n* prime?

- Implementing this in a program:

```
int main() {
    int n{};
    std::cout << "Enter an integer: ";
    std::cin >> n;

    bool is_prime{true};

    for ( int k{2}; k < n; ++k ) {
        if ( n%k == 0 ) {
            is_prime = false;
        }
    }

    if ( is_prime ) {
        std::cout << "The integer " << n << " is prime" << std::endl;
    } else {
        std::cout << "The integer " << n << " is not prime" << std::endl;
    }

    return 0;
}
```

```
for ( int k{2}; k <= n - 1; ++k ) {


}
```

Is *n* prime?

- Do we have to test all integers?
  - If *n* is divisible by 14,
    then *n* must be divisible by at least one of 2 or 7
  - Therefore, we only have to test if *n* is divisible
    by all prime numbers *k* between 2 and *n* – 1
  - Problem: we don't have a list of all prime numbers…
  - We do know, however, that all even numbers after 2 are not prime
    - Can we avoid calculating `n%k` for even values of `k`?

- Strategy: test if `n%2 == 0`,
  if not, test `n%k == 0` for `k` from `3`, `5`, `7`, `9`, …, up to `n - 1`

Is *n* prime?

- We could use the following condition statement and for loop:

```
if ( (n != 2) && (n%2 == 0) ) {
    is_prime = false;
} else {
    for ( int k{3}; k < n; ++k ) {
        // Only test if n is divisible by k for odd k
        if ( k%2 != 0 ) {
            // k must be odd
            if ( n%k == 0 ) {
                is_prime = false;
            }
        }
    }
}
```

Is *n* prime?

- Recall that ++k is the same as `k += 1`, so this is also valid:

```
if ( n%2 == 0 ) {
    is_prime = false;
} else {
    for ( int k{3}; k < n; k += 1 ) {
        // Only test if n is divisible by k for odd k
        if ( k%2 != 0 ) {
            // k must be odd
            if ( n%k == 0 ) {
                is_prime = false;
            }
        }
    }
}
```

Is *n* prime?

- Therefore, we could just use the following:

```
if ( n%2 == 0 ) {
    is_prime = false;
} else {
    for ( int k{3}; k < n; k += 2 ) {
        if ( n%k == 0 ) {
            is_prime = false;
        }
    }
}
```

Is *n* prime?

- Let us determine if an integer *n* is prime

```
bool is_prime{true};

if ( n <= 1 ) {
    is_prime = false;
} else if ( n == 2 ) {
    // Do nothing
} else if ( n%2 == 0 ) {
    is_prime = false;
} else {
    for ( int k{3}; k < n; k += 2 ) {
        if ( n%k == 0 ) {
            is_prime = false;
        }
    }
}

if ( is_prime ) {
    std::cout << "The integer " << n << " is prime" << std::endl;
} else {
    std::cout << "The integer " << n << " is not prime" << std::endl;
}
```

Different update statements

- Here we use this loop variable in a calculation:

```
int sum{0};

for ( int k{1}; k <= 20; k *= 2 ) {
    // The loop body
    sum += k;
}

std::cout << sum << std::endl;
```

k	32
sum	31

Output:

```
31
```

Looping down

- You can also use this loop variable:

```
for ( int k{4}; k >= 0; --k ) {
    // The loop body
    std::cout << k << std::endl;
}
```

Output:

```
4
3
2
1
0
```

  - A loop variable k is initialized with the value

k -1

Arbitrary starting and ending points

- Of course, your end-points need not be `0` or `1`:

```
for ( int k{256}; k < 1024; ++k ) {
    // The loop body
    std::cout << k << std::endl;
}
```

  - Loops with `k` taking values from `256` up to `1023`

```
for ( int k{256}; k > 128; --k ) {
    // The loop body
    std::cout << k << std::endl;
}
```

  - Loops with `k` taking values from `256` down to `129`

The most important loop

- The most important loop you will see in this course:
  - Given any positive integer N

```
for ( int k{0}; k < N; ++k ) {
    // The loop body
    std::cout << k << std::endl;
}
```

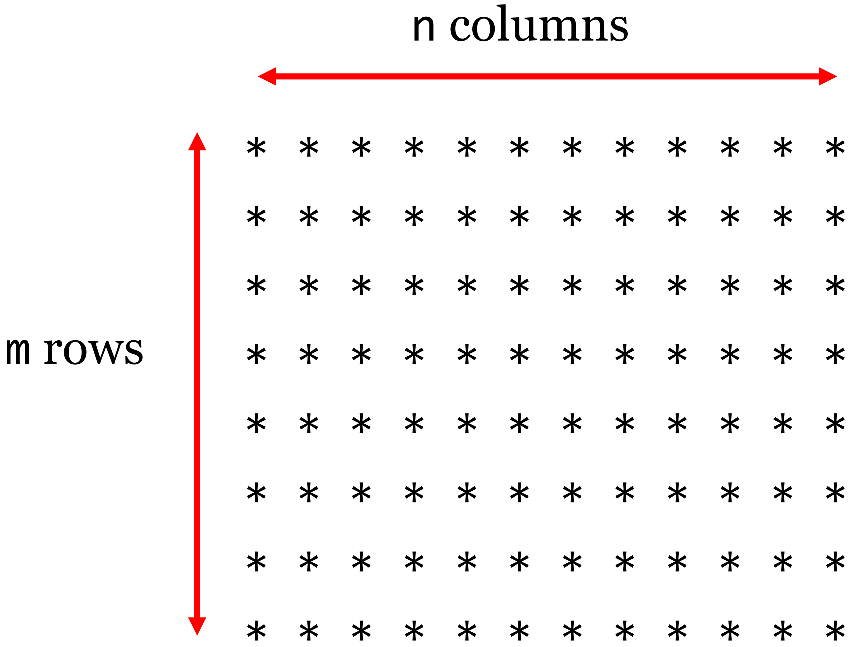
loops with k taking values from 0 up to N - 1

  - This is equivalent to:

```
for ( int k{0}; k != N; ++k ) {
    // The loop body
    std::cout << k << std::endl;
}
```

Loops within loops

- Loops within loops:
  - Given two integers, `m` and `n`, create the following ASCII art:



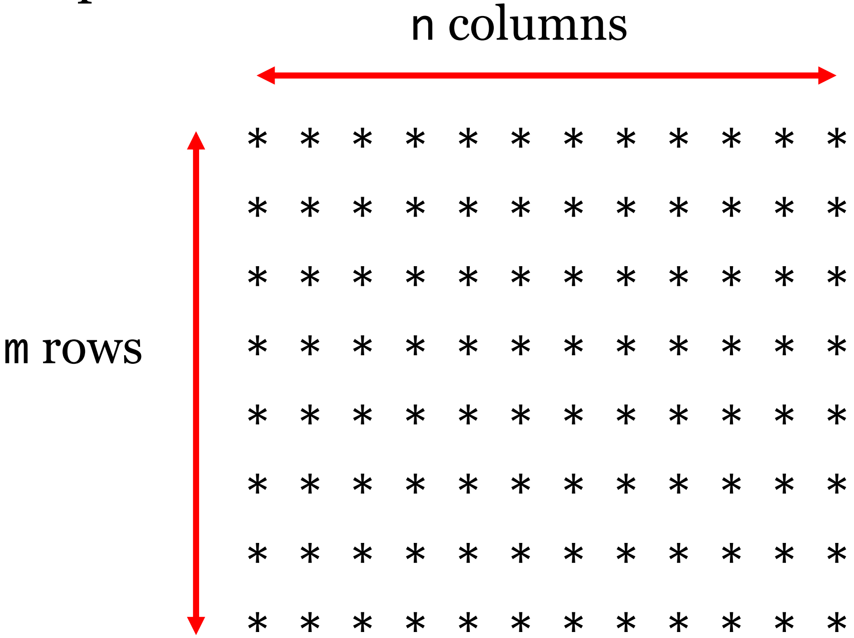
`n` columns

`m` rows

```
*  *  *  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *  *  *  *  *
```

Loops within loops

- We will require a loop that prints each of the `m` rows
  - This outer loop must run from `1` to `m`
- For each row, we must print `n` asterisks
  - This requires an inner loop from `1` to `n`
  - At the end of each execution of the inner loop, we must print an end-of-line



`n` columns

`m` rows

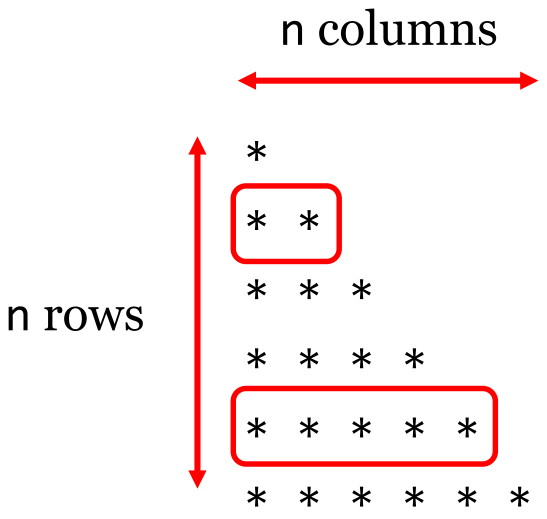
```
*  *  *  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *  *  *  *  *
*  *  *  *  *  *  *  *  *  *  *  *
```

Loops within loops

```
int main() {
    int m{};
    int n{};
    std::cout << "Enter the number of rows: ";
    std::cin >> m;
    std::cout << "Enter the number of columns: ";
    std::cin >> n;

    for ( int rows{1}; rows <= m; ++rows ) {
        for ( int colunms{1}; columns <= n; ++columns ) {
            std::cout << "* ";
        }

        std::cout << std::endl;
    }

    return 0;
}
```

Loops within loops

- Loops within loops:
  - Given one integer, n, create the following ASCII art:



n columns

n rows

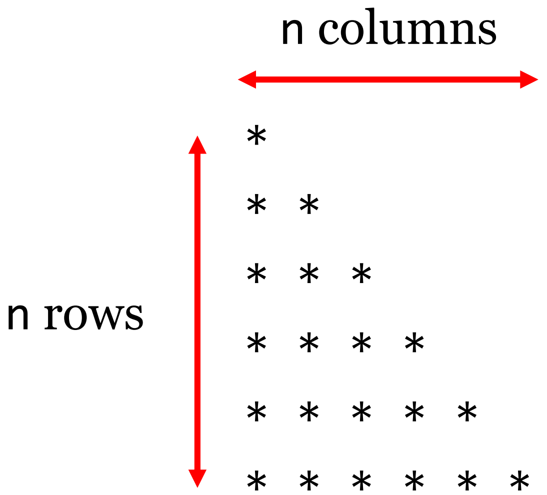
```
*
* *
* * *
* * * *
* * * * *
* * * * * *
```

Loops within loops

```
int main() {
    int n{};
    std::cout << "Enter the number of rows of the square matrix: ";
    std::cin >> n;

    for ( int rows{1}; rows <= n; ++rows ) {
        for ( int colunms{1}; columns <= n; ++columns ) {
            if ( columns <= rows ) {
                std::cout << "* ";
            }
        }

        std::cout << std::endl;
    }

    return 0;
}
```

Loops within loops

- Note, however,
  - In row 1, we print 1 asterisk
  - In row 2, we print 2 asterisks
  - In row 3, we print 3 asterisks



n columns

n rows

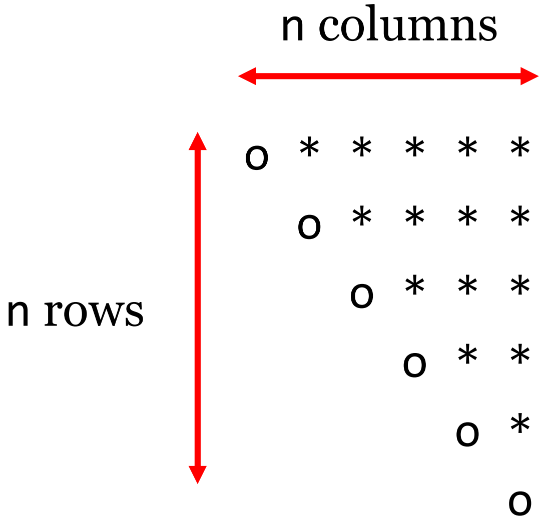
```
*
* *
* * *
* * * *
* * * * *
* * * * * *
```

Loops within loops

```
int main() {
    int n{};
    std::cout << "Enter the number of rows of the square matrix: ";
    std::cin >> n;

    for ( int rows{1}; rows <= n; ++rows ) {
        for ( int colunms{1}; columns <= rows; ++columns ) {
            std::cout << "* ";
        }

        std::cout << std::endl;
    }

    std::cout << std::endl;

    return 0;
}
```

Conditional statements within loops within loops

- Loops within loops:
  - Given one integer, n, create the following ASCII art:



n columns

```
o * * * * *
  o * * * *
    o * * *
      o * *
        o *
          o
```

n rows

Conditional statements within loops within loops

```
for ( int rows{1}; rows <= n; ++rows ) {
    for ( int colunms{1}; columns <= n; ++columns ) {
        if ( columns < rows ) {
            std::cout << "  ";
        } else if ( columns == rows ) {
            std::cout << "o ";
        } else {
            std::cout << "* ";
        }
    }

    std::cout << std::endl;
}

std::cout << std::endl;
```

Applications of loops within loops

- These sound like silly games, but these algorithms are all essential for implementations of linear algebra algorithms
  - Initializing the entries of an $m \times n$ matrix
  - Multiplying an n-dimensional vector by a $m \times n$ matrix
  - Performing Gaussian elimination on a system of n linear equations in n unknowns
  - Using backward substitution to find a solution to such a system in row-echelon form
  - Multiplying an $\ell \times m$ matrix and a $m \times n$ matrix

Summary

- Following this lesson, you now
  - Understand how to construct and run a for loop
  - Know how to use the loop variable within the loop body
  - Understand how we can determine if an integer is prime in C++
    - We will see more efficient algorithms later
  - Know that the initial value, the conditional statement, and the update statement can all be modified as necessary
  - Understand why a loop may be used inside another loop
    - Especially with applications in linear algebra
    - This includes some that require loops within loops within loops
  - Know that the inner loop can also depend on the loop variable of an outer loop

References

[1] Wikipedia
https://en.wikipedia.org/wiki/For_loop

[2] cplusplus.com
http://www.cplusplus.com/doc/tutorial/control/

Acknowledgments

Proof read by Dr. Thomas McConkey and Charlie Liu.

Colophon

These slides were prepared using the Georgia typeface. Mathematical equations use Times New Roman, and source code is presented using `Consolas`.

The photographs of lilacs in bloom appearing on the title slide and accenting the top of each other slide were taken at the Royal Botanical Gardens on May 27, 2018 by Douglas Wilhelm Harder. Please see

https://www.rbg.ca/

for more information.

Disclaimer

These slides are provided for the ECE 150 *Fundamentals of Programming* course taught at the University of Waterloo. The material in it reflects the authors' best judgment in light of the information available to them at the time of preparation. Any reliance on these course slides by any party for any other purpose are the responsibility of such parties. The authors accept no responsibility for damages, if any, suffered by any party as a result of decisions made or actions based on these course slides for any other purpose than that for which it was intended.