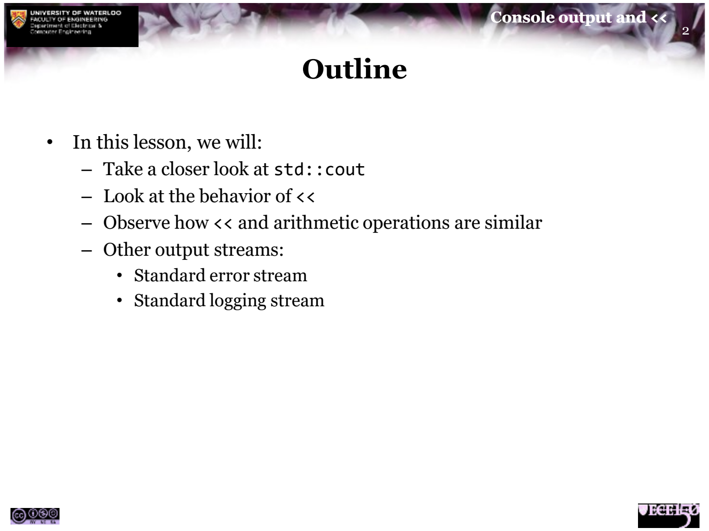
UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical & Computer Engineering

ECE 150 *Fundamentals of Programming*

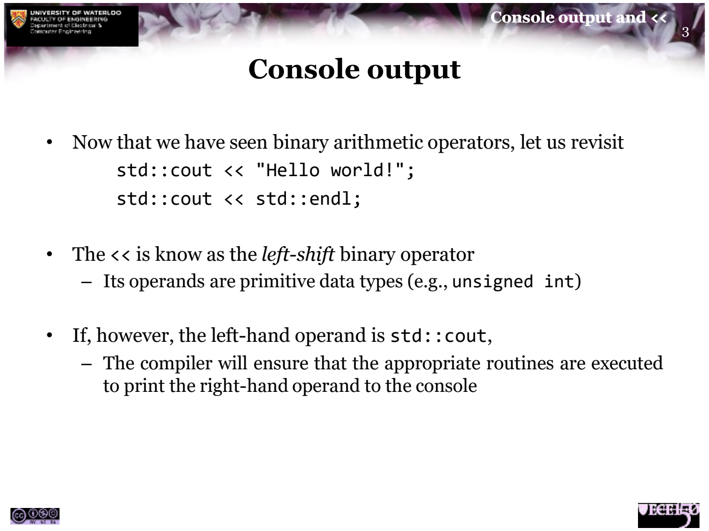
Console output

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



Outline

- In this lesson, we will:
  - Take a closer look at `std::cout`
  - Look at the behavior of `<<`
  - Observe how `<<` and arithmetic operations are similar
  - Other output streams:
    - Standard error stream
    - Standard logging stream

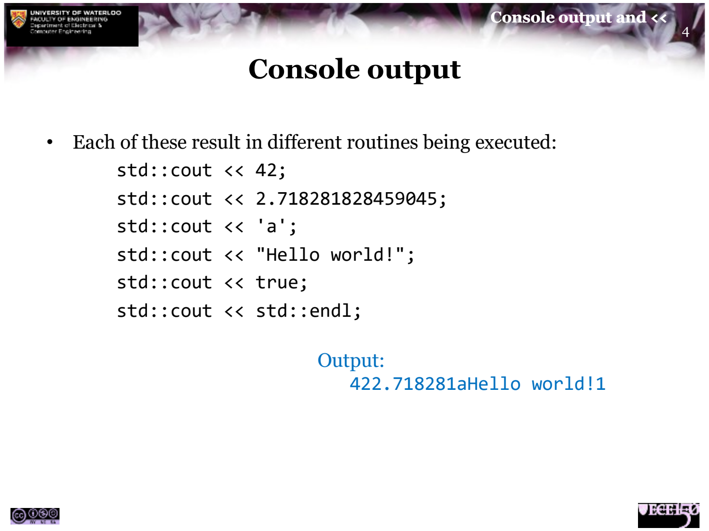


Console output

- Now that we have seen binary arithmetic operators, let us revisit

```
std::cout << "Hello world!";
std::cout << std::endl;
```

- The `<<` is know as the *left-shift* binary operator
  - Its operands are primitive data types (e.g., `unsigned int`)
- If, however, the left-hand operand is `std::cout`,
  - The compiler will ensure that the appropriate routines are executed to print the right-hand operand to the console



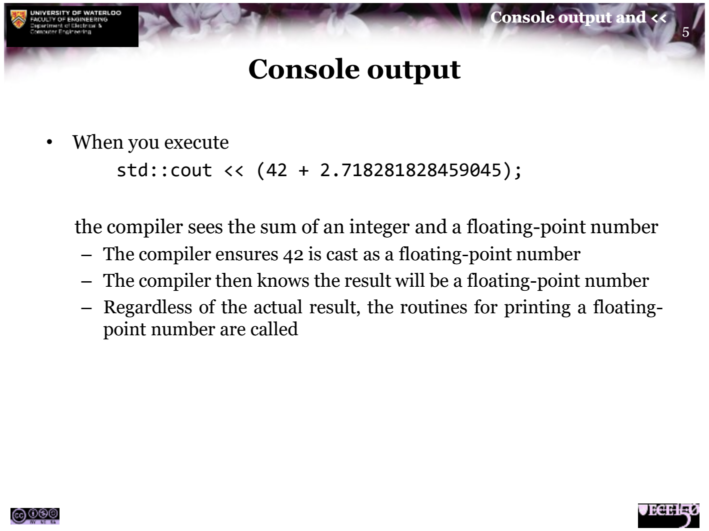
Console output

- Each of these result in different routines being executed:

```
std::cout << 42;
std::cout << 2.718281828459045;
std::cout << 'a';
std::cout << "Hello world!";
std::cout << true;
std::cout << std::endl;
```

Output:

```
422.718281aHello world!1
```

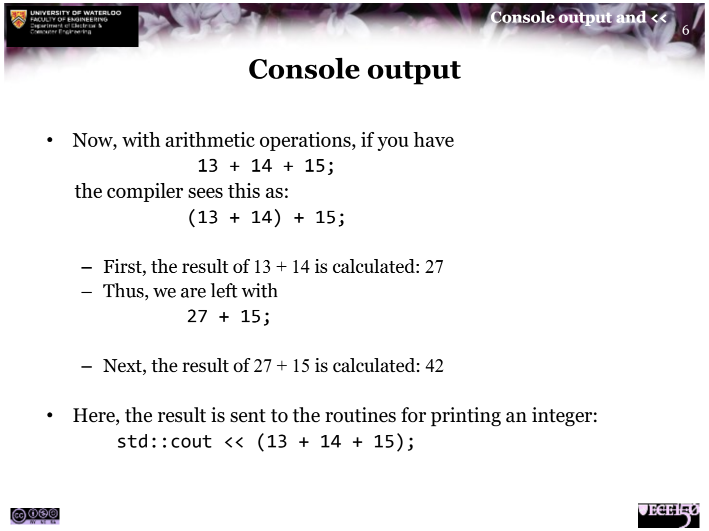


Console output

- When you execute

```
std::cout << (42 + 2.718281828459045);
```

  the compiler sees the sum of an integer and a floating-point number
  - The compiler ensures 42 is cast as a floating-point number
  - The compiler then knows the result will be a floating-point number
  - Regardless of the actual result, the routines for printing a floating-point number are called



Console output

- Now, with arithmetic operations, if you have

```
13 + 14 + 15;
```

  the compiler sees this as:

```
(13 + 14) + 15;
```

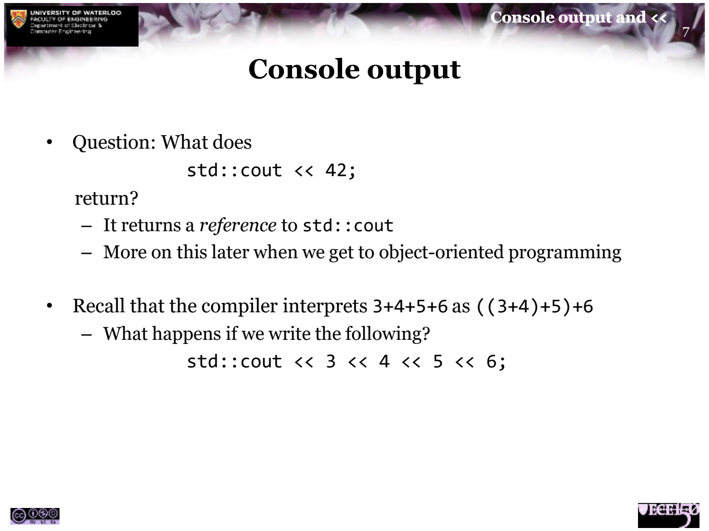
  - First, the result of 13 + 14 is calculated: 27
  - Thus, we are left with

```
27 + 15;
```

  - Next, the result of 27 + 15 is calculated: 42

- Here, the result is sent to the routines for printing an integer:

```
std::cout << (13 + 14 + 15);
```



Console output

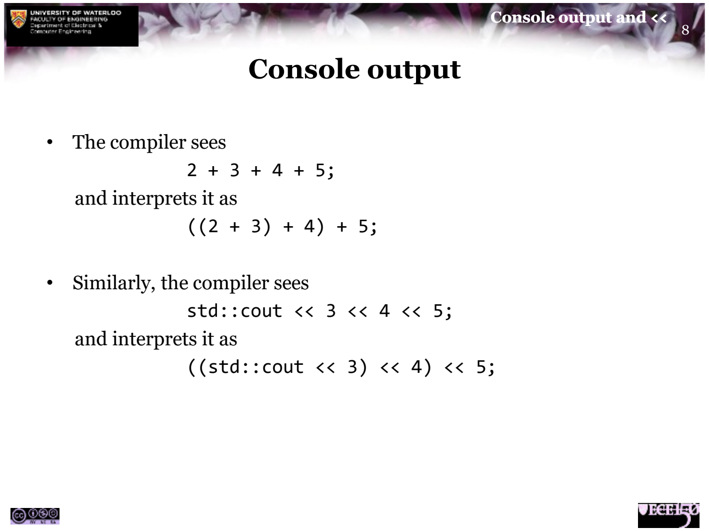
- Question: What does

```
std::cout << 42;
```

  return?
  - It returns a *reference* to `std::cout`
  - More on this later when we get to object-oriented programming

- Recall that the compiler interprets `3+4+5+6` as `((3+4)+5)+6`
  - What happens if we write the following?

```
std::cout << 3 << 4 << 5 << 6;
```



Console output

- The compiler sees

```
2 + 3 + 4 + 5;
```

  and interprets it as

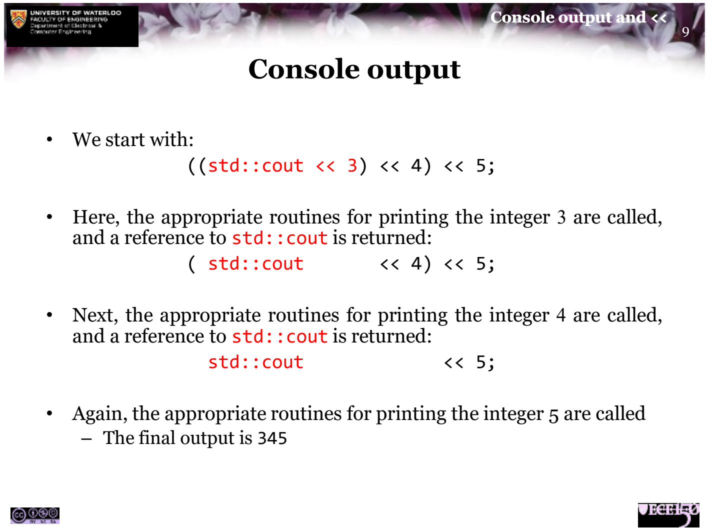
```
((2 + 3) + 4) + 5;
```

- Similarly, the compiler sees

```
std::cout << 3 << 4 << 5;
```

  and interprets it as

```
((std::cout << 3) << 4) << 5;
```



Console output

- We start with:

```
((std::cout << 3) << 4) << 5;
```

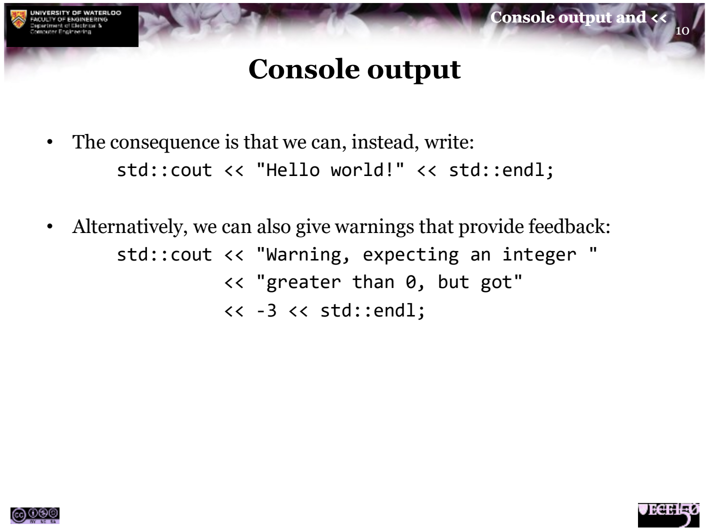
- Here, the appropriate routines for printing the integer 3 are called, and a reference to `std::cout` is returned:

```
( std::cout       << 4) << 5;
```

- Next, the appropriate routines for printing the integer 4 are called, and a reference to `std::cout` is returned:

```
  std::cout             << 5;
```

- Again, the appropriate routines for printing the integer 5 are called
  - The final output is 345



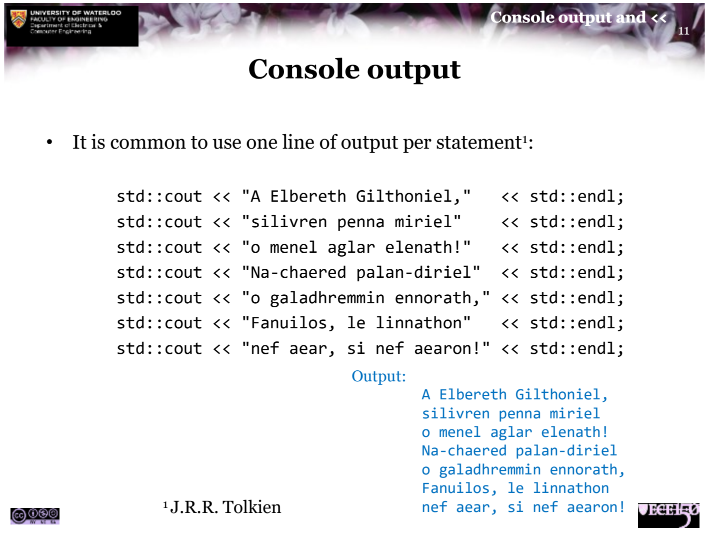
Console output

- The consequence is that we can, instead, write:

```
std::cout << "Hello world!" << std::endl;
```

- Alternatively, we can also give warnings that provide feedback:

```
std::cout << "Warning, expecting an integer "
          << "greater than 0, but got"
          << -3 << std::endl;
```



Console output

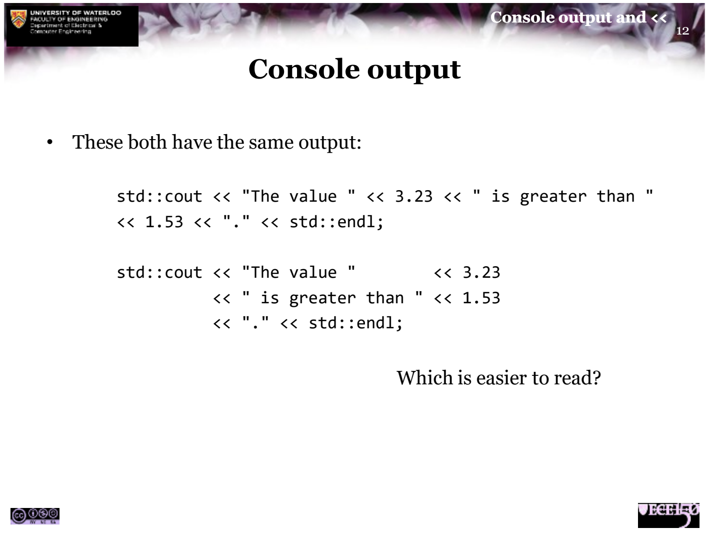
- It is common to use one line of output per statement[1]:

```
std::cout << "A Elbereth Gilthoniel,"   << std::endl;
std::cout << "silivren penna miriel"    << std::endl;
std::cout << "o menel aglar elenath!"   << std::endl;
std::cout << "Na-chaered palan-diriel"  << std::endl;
std::cout << "o galadhremmin ennorath," << std::endl;
std::cout << "Fanuilos, le linnathon"   << std::endl;
std::cout << "nef aear, si nef aearon!" << std::endl;
```

Output:

```
A Elbereth Gilthoniel,
silivren penna miriel
o menel aglar elenath!
Na-chaered palan-diriel
o galadhremmin ennorath,
Fanuilos, le linnathon
nef aear, si nef aearon!
```

[1] J.R.R. Tolkien



Console output

- These both have the same output:

```
std::cout << "The value " << 3.23 << " is greater than "
<< 1.53 << "." << std::endl;

std::cout << "The value "        << 3.23
          << " is greater than " << 1.53
          << "." << std::endl;
```

Which is easier to read?

Other output streams

- In addition to `std::cout`, there are two other output streams:
  - Standard error stream:

    ```
    std::cerr << "Division by zero..." << std::endl;
    ```
  - Standard logging stream:

    ```
    std::clog << "Current size:" << 3 << std::endl;
    ```

- By default, all streams go to the output console
  - It is possible to redirect each stream to different outputs, e.g.,
    - All standard output goes to the console
    - All logging output goes to a log file
    - All error output goes to an error file

Summary

- Following this lesson, you now:
  - Understand that `<<` is a binary operator and `std::cout` is a stream object
  - Know the result of `std::cout << `*`whatever`* is a reference to itself, namely `std::cout`
  - Know that we can and why we can string outputs to `std::cout` together
  - Are aware of standard error streams and standard logging streams

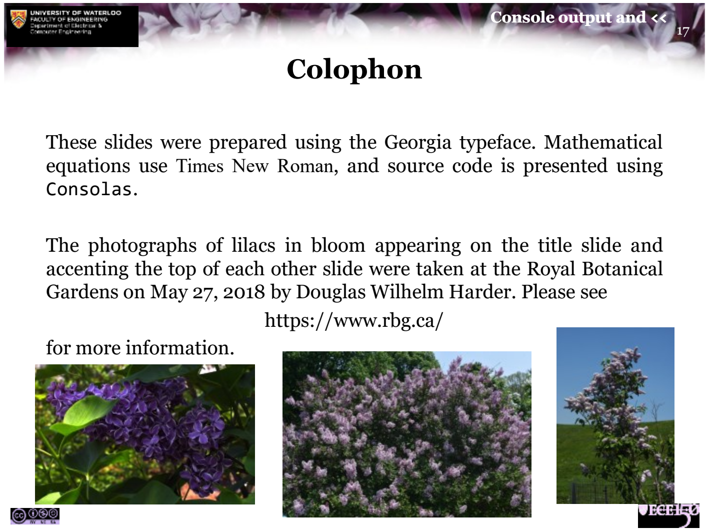
References

[1] cplusplus.com
http://www.cplusplus.com/doc/tutorial/basic_io/
http://www.cplusplus.com/reference/iostream/cout/

[2] Wikipedia
https://en.wikipedia.org/wiki/Input/output_(C++)

Acknowledgments

Proof read by Dr. Thomas McConkey



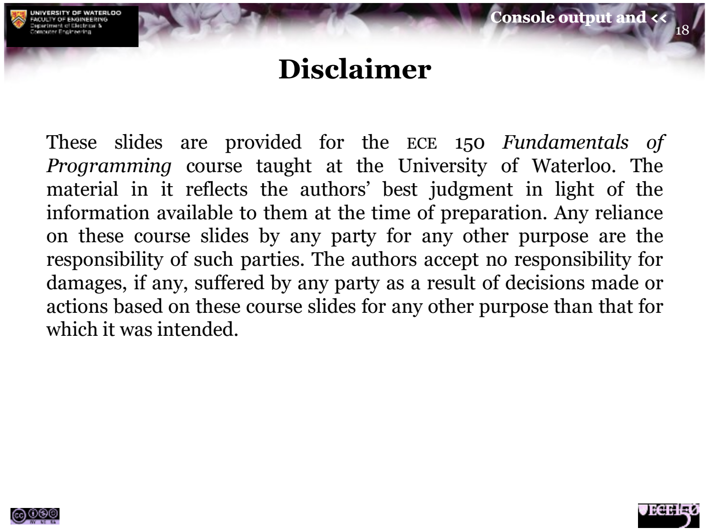
Colophon

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.