

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

ECE 150 Fundamentals of Programming

Literal data

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

© 2018 by Douglas Wilhelm Harder and Hiren Patel.  
Some rights reserved.

ECE150  
CC BY NC SA

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

Literal data

Outline

- In this presentation, we will:
  - Define literal data
  - Describe:
    - Integers
    - Characters
    - Strings
    - Floating-point numbers (reals)
    - Boolean

ECE150

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

Literal data

Literals

- Often we must hard-code data into our programs
- Such data are called *literals*—they are literally what they represent
- We have seen:
  - The integer `0`
  - A literal phrase of text `"Hello world!"`
- There are five categories of literal data:
  - Integers
  - Characters
  - Strings
  - Floating-point numbers
  - Boolean

ECE150

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

Literal data

Integer literals

- We have seen an integer literal  
`return 0;`
- Any sequence of decimal digits not starting with a `0` is interpreted as an integer literal, possibly prefixed with either `+` or `-`

<code>-0</code>	<code>0</code>	<code>+0</code>
<code>-42</code>	<code>42</code>	<code>+42</code>
<code>-1023</code>	<code>1023</code>	<code>+1023</code>
<code>-1048576</code>	<code>1048576</code>	<code>+1048576</code>

ECE150

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

Literal data 5

## Integer literals

```
#include <iostream>

int main();

int main() {
    std::cout << "The answer to the ultimate question is ";
    std::cout << 42;
    std::cout << std::endl;

    return 0;
}
```

Output:  
The answer to the ultimate question is 42



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

Literal data 6

## Integer literals

- Instead of describing such representations in words, there is a more concise format in computer science: regular expressions:

$[+-]?[1-9][0-9]^*$

Optionally a '+' or a '-'

Zero or more additional digits from '0' to '9'

Required: a digit from '1' to '9'

$[+-]?0$

Optionally a '+' or a '-'

Required: the digit '0'



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

Literal data 7

## Integer literals

- We can combine these in one of two ways:
  - The first or the second:

$[+-]?0|[+-]?[1-9][0-9]^*$

- An optional + or – followed by a 0 or a non-zero integer:

$[+-]?(0|[1-9][0-9]^*)$

... or ...



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

Literal data 8

## Character literals

- Books are a sequence of letters, numbers or punctuation
- All of these symbols are collectively called *characters*
- There are two common representations of characters:
  - ASCII<sup>1</sup>
  - Unicode

<sup>1</sup> American Standard Code for Information Interchange





## Character literals

- ASCII is limited to 128 characters stored in one byte
  - 33 code for non-printing control characters (e.g., TAB, BS, CR, BELL)

```
BLANK! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
@ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [ \ ] ^ _
` a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~ DEL
```

- Most keyboards include all 95 printable ASCII characters and some control characters



The BLANK TAB NL characters are collectively called *whitespace* characters



Reference: <https://en.wikipedia.org/wiki/ASCII>



## Character literals

- Unicode is designed to encode most writing systems
  - Unicode 11.0:
    - contains 137,439 characters
    - covers 146 modern and historic scripts
    - symbol sets and emoji

π 𐀀 𐀁 𐀂 𐀃 𐀄



## Character literals

- Printable characters (those on a keyboard) can be literally encoded in C++ source code by using single quotes:

```
#include <iostream>

int main();

int main() {
    std::cout << 'a';
    std::cout << 'b';
    std::cout << 'c';
    std::cout << std::endl;

    return 0;
}
```

Output:  
abc



## Escape sequences

- Problem: How do you store a literal single quote?
- Solution: *escape sequences*
  - An *escape character* indicates that the next character is interpreted
    - For C++ characters, the escape character is \
  - The compiler sees '\'' but treats it as the single ASCII character for an apostrophe
    - A literal backslash `'\\'`
    - The TAB character `'\t'`



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

Literal data 13

## End-of-line characters

- The ASCII representation was designed for teletype machines
  - Automated typewriters
  - The carriage return `CR` control character (`'\r'`) moved the printing carriage back to the start of the line
  - The line feed `LF` control character (`'\n'`) rotated the roller for the next line
    - The *new-line* character
- You needed to send both characters: `CR LF`



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

Literal data 14

## End-of-line characters

- Computer screens automatically go to the start of the next line
  - Unix (and now Linux and macOS) chose `LF`
  - The classic Mac OS chose `CR`
  - Microsoft DOS kept both: `CR LF`
- This causes compatibility and portability issues...



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

Literal data 15

## End-of-line characters

- In C, your code depends on the platform:
 

```
printf( "Hello world!\n" ); // Unix/Linux/macOS
printf( "Hello world!\r" ); // classic Mac OS
printf( "Hello world!\r\n" ); // Microsoft
```
- In C++, the compiler deals with it:
 

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



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

Literal data 16

## String literals

- A sequence of characters is described as a *string of characters*
  - More simply, a *string*
- When we include "Hello world!" directly in our source code, we call this a *string literal*
  - That is literally the string to be used
- A string encompasses all characters after the opening double quote up to the closing double quote, which **must** be on the same line
- A string with a single character is not a character:
  - `'a'` and `"a"` are different
  - The first is a character, the second is a string with one character...



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

Literal data 17

## Escape sequences

- The escape sequence for C++ strings is also the backslash:

```
std::cout << "She said \"Hello world!\"";
```

```
She said "Hello world!"
```

```
std::cout << "Look in C:\\Users\\dwharder";
```

```
Look in C:\Users\dwharder
```

```
std::cout << "Times:\t0.1 s\t23.4 s\t56.789 s\t0 s";
```

```
Times:  0.1 s   23.4 s   56.789 s   0 s
        |       |       |       |
        Tab stops
```



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

Literal data 18

## Escape sequences

- The escape sequence is only for encoding
  - Internally, "She said \"Hello world!\"" is stored as 23 characters
- Escape sequences are common in computer programming:
  - The Extended Markup Language (XML) uses & followed by ;
    - &gt; codes the greater-than symbol (>)
    - &infin; codes the infinity symbol ( $\infty$ )
    - &amp; codes the ampersand (&)



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

Literal data 19

## Floating-point literals

- We cannot store real numbers to arbitrary precision
  - $\pi$  to 769 digits of precision:

```
3.1415926535897932384626433832795028841971693993751058209749445
923078164062862089986280348253421170679821480865132823066470938446
0955058223172535940812848111745028410270193852110555964462294895493038
19644288109756659334461284756482337867831652712019091456485669234603486104
5432664821339360726024914127372458700660631558817488152092096282925409171536436
789259036001133053054882046652138414695194151160943305727036575959195309218611738193
261179310511854807446237996274956735188575272489122793818301194912983367336244065664
308602139494639522473719070217986094370277053921717629317675238467481846766940513200
056812714526356082778577134275778960917363717872146844090122495343014654958537105079
2279689258923542019956112129021960864034418159813629774771309960518707211349999998...
```

- We can only store a finite number of digits of precision relative to a decimal point
  - We call such representations *floating-point*



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

Literal data 20

## Floating-point literals

- Any sequence of decimal digits that has a decimal point (period) somewhere is considered a floating-point literal
  - Can be prefixed by either a + or -
 

- .42	4.2	+42.
+ .1023	-10.23	1023.
.1048576	+1048.576	-1048576.
  - Other representations are possible—we will discuss these later



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

Literal data 21

## Floating-point literals

- Printing floating-point numbers is different printing other literals:

```
#include <iostream>

int main();

int main() {
    std::cout << "Some floats: ";
    std::cout << std::endl;
    std::cout << 3.0;
    std::cout << std::endl;
    std::cout << 3.14;
    std::cout << std::endl;
    std::cout << 3.141592653589793;
    std::cout << std::endl;

    return 0;
}
```

Output:  
Some floats:  
3  
3.14  
3.14159



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

Literal data 22

## Floating-point literals

- We can describe floating-point numbers using regular expressions:

this or this

$$[+-]?[0-9]+\.\.[0-9]* \quad [+-]?[0-9]*\.\.[0-9]+$$

[+-]?      Optionally a '+' or a '-'  
[0-9]+    One or more digits from '0' to '9'  
[0-9]\*    Zero or more digits from '0' to '9'  
\.        A period (decimal point), as . is special  
          – Yes, regular expressions use escape characters, too



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

Literal data 23

## Boolean literals

- The last category of literal in C++ are Boolean literals:

```
true|false

#include <iostream>

int main();

int main() {
    std::cout << true;
    std::cout << std::endl;
    std::cout << false;
    std::cout << std::endl;

    return 0;
}
```

Output:  
1  
0



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

Literal data 24

## Summary of regular expressions

- We are only introducing some aspects of regular expressions in this lecture, and the variations on regular expressions differ
  - Most differ by what needs to be escaped where
- For more details, please see [https://ece.uwaterloo.ca/~ece150/Lecture\\_materials/1.03/Regular%20expressions.pdf](https://ece.uwaterloo.ca/~ece150/Lecture_materials/1.03/Regular%20expressions.pdf)





## Sample code

- Examples of literal data are shown here:  
<https://repl.it/@dwharder/Literal-data>



## References

- [1] Wikipedia  
[https://en.wikipedia.org/wiki/Literal\\_\(computer\\_programming\)](https://en.wikipedia.org/wiki/Literal_(computer_programming))
- [2] cplusplus.com tutorial  
<http://www.cplusplus.com/doc/tutorial/constants/>
- [3] C++ reference  
[https://en.cppreference.com/w/cpp/language/integer\\_literal](https://en.cppreference.com/w/cpp/language/integer_literal)  
[https://en.cppreference.com/w/c/language/integer\\_constant](https://en.cppreference.com/w/c/language/integer_constant)



## Summary

- After this lesson, you now
  - Understand the idea of literal data in source code
  - You understand how to encode
    - Integers
    - Characters
    - Strings
    - Floating-point numbers (reals)
    - Boolean
 in your source code
  - Everything else in C++ deals with the storage and manipulation of data



## Acknowledgments

Proof read by Dr. Thomas McConkey





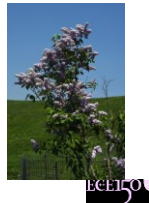
## Colophon

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.

