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

ECE 150 *Fundamentals of Programming*

Literal data

Prof. Hiren Patel, Ph.D. P.Eng.
Prof. Werner Dietl, Ph.D.
Prof. Douglas Wilhelm Harder, M.Math. LEL

Outline

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

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

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 `-`

```
#include <iostream>

int main();
```

-0	0	+0
-42	42	+42
-1023	1023	+1023
-1048576	1048576	+1048576

```
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
```

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[1]
  - Unicode

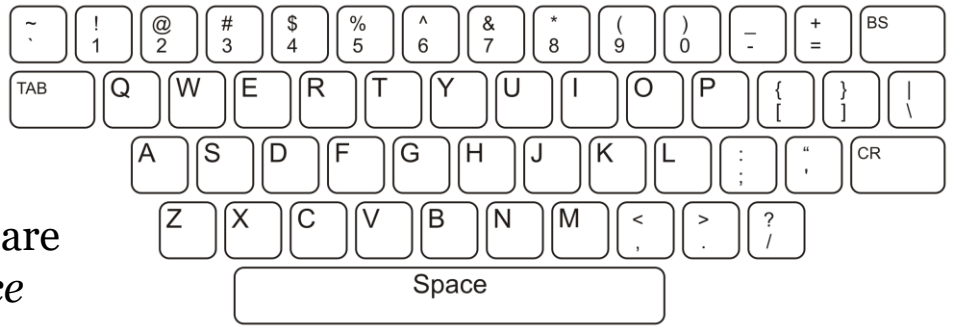
[1] American Standard Code for Information Interchange

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 Я 音 æ∞

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;
}
```

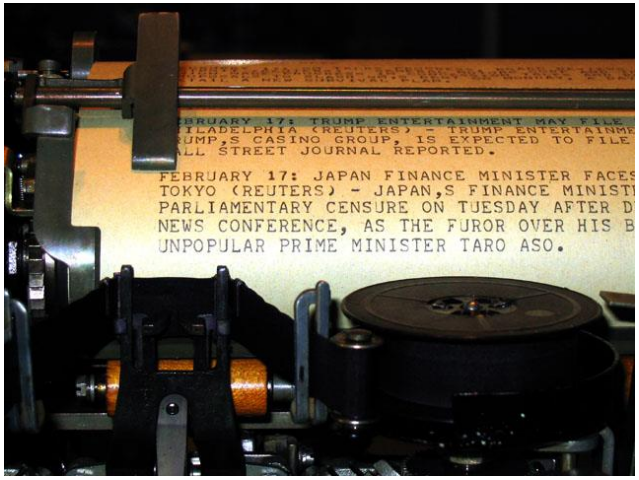
Output:
abc

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'`

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**

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: CRLF

- This causes compatibility and portability issues…
  - 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;
```

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
- A string with no characters is valid: `""`
  - `''` is nonsensical: a character must be a character

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

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), *tags* use < and >

    ```
    <p>Hello world!</p>
    ```
  - How do you print < or > or non-ascii characters?
  - XML uses & followed by ;
    - `>` codes the greater-than symbol (>)
    - `&infin;` codes the infinity symbol (∞)
    - `&` codes the ampersand (&)

Floating-point literals

- We cannot store real numbers to arbitrary precision
  - π to 769 digits of precision:

3.14159265358979323846264338327950288419716939937510582097494459
23078164062862089986280348253421170679821480865132823066470938446
09550582231725359408128481117450284102701938521105559644622948954930381
96442881097566593344612847564823378678316527120190914564856692346034861045
43266482133936072602491412737245870066063155881748815209209628292540917153643678
92590360011330530548820466521384146951941511609433057270365759591953092186117381932
61179310511854807446237996274956735188575272489122793818301194912983367336244065664
30860213949463952247371907021798609437027705392171762931767523846748184676694051320005
68127145263560827785771342757789609173637178721468440901224953430146549585371050792279
68925892354201995611212902196086403441815981362977477130996051870721134999999839

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 -

```
-0.42          4.2          +42.0
+0.1023        -10.23       1023.0
 0.1048576   +1048.576   -1048576.0
```

  - Other representations are possible—we will discuss these later

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
```

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
```

Sample code

- Examples of literal data are shown here:

  `https://replit.com/@dwharder/Literal-data`

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

References

[1] Wikipedia
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/c/language/integer_constant

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.