Basic data types

Building blocks of computation
Goals

• By the end of this lesson you will be able to:
  - Understand the commonly used basic data types of C++ including
    • Characters
    • Integers
    • Floating-point values
    • Boolean types
  - Understand some basics about modifiers and more complicated data types
What are data types?

• Programming typically involves lots of data manipulation such as calculations

• At the lowest level, in a computer everything is represented in **binary** – a sequence of 0’s and 1’s

• It’s impractical to use binary all the time, so programming languages implement **data types** which are more human-friendly representations of data
What are basic data types?

- **Basic data types** refer to the core data types that exist in the language without any additions.
- Basic data types may also be called *primitives* or *fundamental data types*.
- The common basic data types we will use in C++ are:
  - `int` – for integers such as -288, 0, or 39791
  - `double` – for real numbers such as -30.5, 0.0001, or 2737.118
  - `char` – for single characters such as Q, 5, or &
  - `boolean` – for logical truth values such as true or false
Understanding ranges

- Every data type has a possible range of values that restricts the values that can be stored within it.
- These ranges are based on the fact that data takes up memory.
- For example, it takes little memory to represent/store a 3-digit integer, but considerably more for a 10,000 digit integer.
- Data types have a limit on the number of bytes they can take up, and this dictates the range of values.
Recall that I mentioned that everything is, at the lowest level, represented as 1’s and 0’s.

Let’s think about what an integer actually is, for example 416.

416 contains three digits but what do they mean?

- 4 is the number of 100’s
- 1 is the number of 10’s
- 6 is the number of 1’s

We can write 416 as $4 \times 100 + 1 \times 10 + 6 \times 1$ or $4 \times 10^2 + 1 \times 10^1 + 6 \times 10^0$. 

---

**Integer encoding**
Binary encoding

• What if, instead of using 1, 10, 100, 1000, etc. as our progression, we used 1, 2, 4, 8, etc.

• Instead of using digits 0 – 9 and multiplying by 10 (decimal), we will use digits 0 – 1 and multiply by 2 (binary)

• The binary number 1101 could thus be converted to decimal as: $1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8 + 4 + 0 + 1 = 13$

• Mathematical operations, such as addition and multiplication, work in the same way (e.g. decimal addition “carries” when greater than 10, binary addition “carries” when greater than 2)
Integer ranges

• The smallest element of memory, a single 0 or 1, in computers is referred to as a **bit** (binary digit)

• A **byte** consists of 8 bits, therefore a byte can take on any one of 256 values (00000000 = 0, 11111111 = 255)

• Note, as well, $256 = 2^8$ – each bit can take on one of two possible values (0 or 1) and there are 8 bits

• Although each implementation may be different, in most C++ implementations an integer has a maximum limit of 4 bytes

• 4 bytes $= 2^{(8 \times 4)} = 2^{32} = 4,294,967,296$ possible values
Integer ranges

- An integer in C++ cannot typically take on values from 0 to 4,294,967,295 because we need to think about negative values as well!
- Instead, the first bit is used to indicate the sign (positive or negative), thus leaving 31 bits to store values
- Also, note that there is no need for both a positive and negative zero, since +0 and -0 both mean the same thing
- Therefore, the range of a typical integer, also called an int in C++, is $-2^{31}$ to $2^{31} - 1 = -2,147,483,648$ to $2,147,483,647$
  - Note: Do not write commas in numbers when writing a program
- For more information about negative integer representation research two’s complement
Real numbers

- Real numbers have a decimal part, so it would make sense that if we only have four bytes to represent the number, the range would be much smaller.

- We opt for a tradeoff in which precision is sacrificed for range.

- For example, if we wish to store a number like 5.10821770310173901838883828913818 it may be impossible, but if we sacrifice the least significant decimal places the result may still be acceptable.
Real numbers

- Note that a real number like -0.000005838 can also be written in scientific notation as \(-5.838 \times 10^{-6}\) and we can break this down into three parts:
  - The **sign** (positive or negative)
  - The **mantissa** (5.838 in this case)
  - The **exponent** (if we always use powers of 10, this exponent is simply -6)

- Likewise, a very large number such as 129048124703971097 could be written as \(1.29048124703971097 \times 10^{17}\)

- Real numbers are also called **floating-point** because the decimal point “floats” within the mantissa for various exponents
Real number encoding

- C++ offers two data types for representing real numbers in the sign/mantissa/exponent format:
  - float (floating-point) – 4 bytes (32 bits)
    - 1 sign bit
    - 8 exponent bits
    - 23 mantissa bits
  - double (double precision) – 8 bytes (64 bits)
    - 1 sign bit
    - 11 exponent bits
    - 52 mantissa bits
The range of a float is $\pm 3.4 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$ with approximately 7-decimal place accuracy.

The range of a double is $\pm 1.7 \times 10^{-308}$ to $\pm 1.7 \times 10^{308}$ with approximately 15-decimal place accuracy.

We will typically use the double data type because it is more accurate and memory on modern computers is not a huge concern.

All integer values consisting of 15 digits or fewer can be stored as a double with no loss of accuracy, though for many reasons we will use an int instead of a double when we are using integer values we know to be small.
Character encoding

- Characters are most commonly the keys that you can hit on your keyboard and consist of letters, numbers, and symbols.
- Again, since computers use binary, there needs to be some mapping from a characters like Q or $ to a sequence of bits.
- A very common character encoding standard is known as the American Standard Code for Information Interchange (ASCII).
An ASCII table provides a standardized mapping of characters to 7 bits, for example:

- A is encoded as 1000001 (65 decimal)
- a is encoded as 1100001 (97 decimal)
- % is encoded as 0100101 (37 decimal)
- 4 is encoded as 0110100 (52 decimal)
- space is encoded as 0100000 (32 decimal)

7 bits allows for $2^7 = 128$ different values

No need for positive or negative, so the range can simply be from 0 to 127
Note that A is a distinct character from a, and each has a different encoding.

Investigation of an ASCII table will also show many interesting things:

- The beginning of the table consists of control characters which are non-visible characters to denote things like the end of a file.
- Digits are collated.
- Uppercase letters are collated.
- Lowercase letters are collated.
  - Mathematically, if you add the number 1 to the character P, you get the character Q.
The ASCII table uses 7-bit encoding but a byte is typically the smallest addressable part of memory and consists of 8 bits.

A character, called a `char` in C++, uses 1 byte and can take on more values than in the ASCII table, so the values outside of the range 0 – 127 are implementation-dependent and are very rarely used anyway.

There is a lot of history behind the ASCII table and other character encoding standards (EBCDIC, Unicode, UTF-8, etc.).
To distinguish the character 4 (binary 110100) from the integer 4 (binary 100), and in fact to denote all char values, we enclose them in single quotes: ' (e.g. ' 4 ', ' $ ', ' z ')

Revisiting our addition idea from before:

- \[ 4 + 4 = 8 \]
- \[ '4' + 4 = 52 + 4 = 56 = '8' \]
- \[ '4' + '4' = 52 + 52 = 104 = 'h' \]
Boolean types

- Boolean data types are used to denote two different values: true and false
- These will be commonly used in programming when we’re using logic to decide whether or not to run a portion of a program (e.g. run this portion if the value is true)
- Because only two values can be represented, we only need a single bit (0 or 1)
bool data type

- C++ utilizes a data type called `bool` which can take on the values `true` or `false`

- `true` is encoded as the bit 1 (decimal 1) and `false` is encoded as the bit 0 (decimal 0)

- Again, because the byte is the smallest addressable memory, we will still use 1 byte (8 bits) to store a `bool`

- As we will see later when examining logical statements, any non-zero value will indicate `true`
Modifiers and other data types

- Modifiers can be used with many data types to modify their size in bytes and/or their representation:
  - `short` – reduces the # of bytes (e.g. `short int`)
  - `long` – increase the # of bytes (e.g. `long double`)
  - `unsigned` – do not use the sign bit (e.g. `unsigned long int`)

- There are other data types we will encounter and create that are built on these basic data types, including strings, arrays, structures, classes, and various abstract data types such as stacks
Exploration

- Practice converting decimal to binary and vice versa
- Examine two’s complement and learn more about why it is used
- Find examples of how floating-point numbers are represented in binary
- Investigate the ASCII table and the history behind it
- Learn about data type modifiers and their ranges