

Datatypes and Variables: More Details

Principles of Computer Programming I
Spring/Fall 20XX



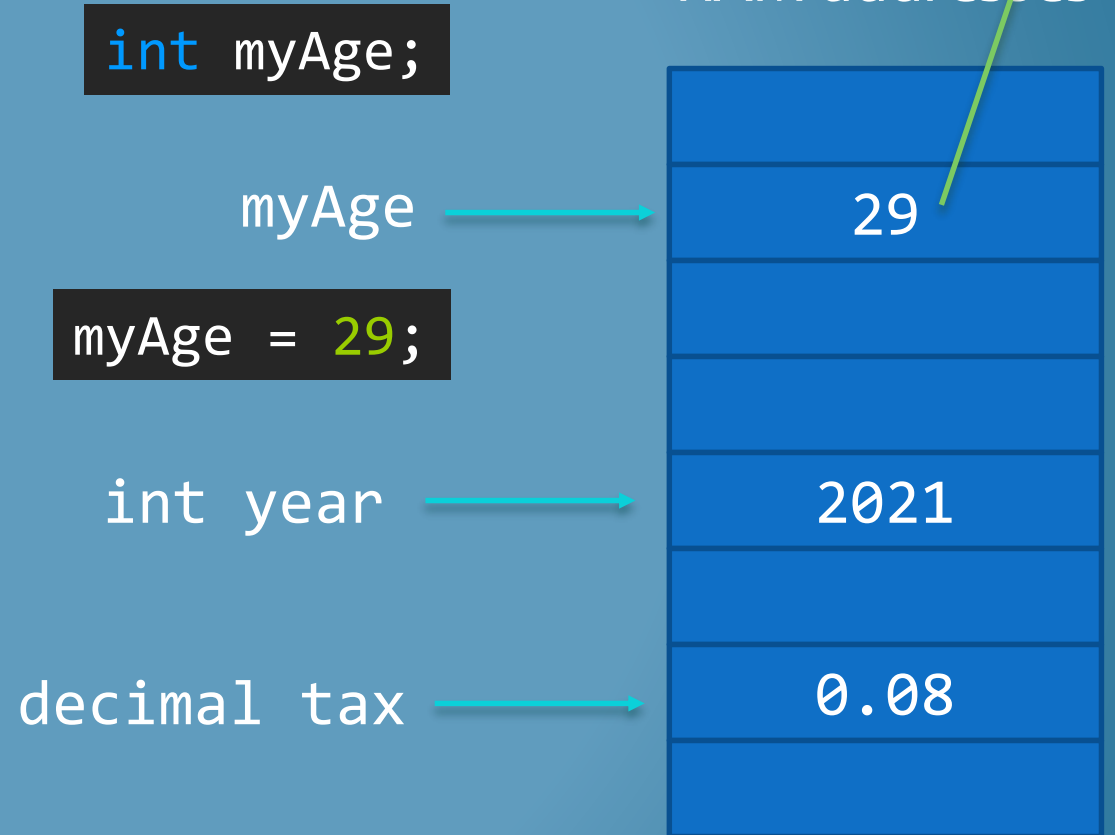
AUGUSTA
UNIVERSITY

Outline

- More About C# Datatypes
 - Integer types and sizes
 - Floating-point type precision
 - Value vs. Reference Types

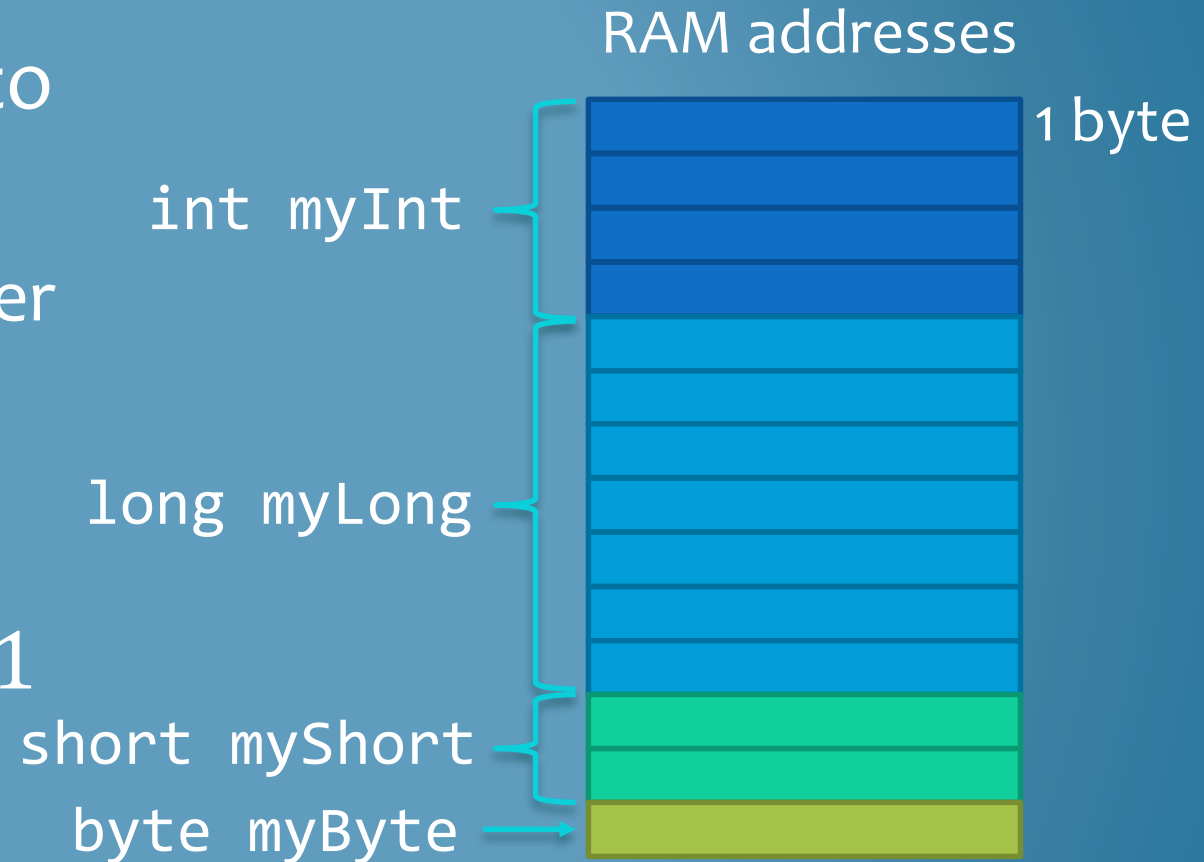
Variables: Memory Locations

- A variable names a *memory location* in which to store data
- Declaring a variable = reserving memory, giving it a name
- Assigning a variable = storing data to that address in memory



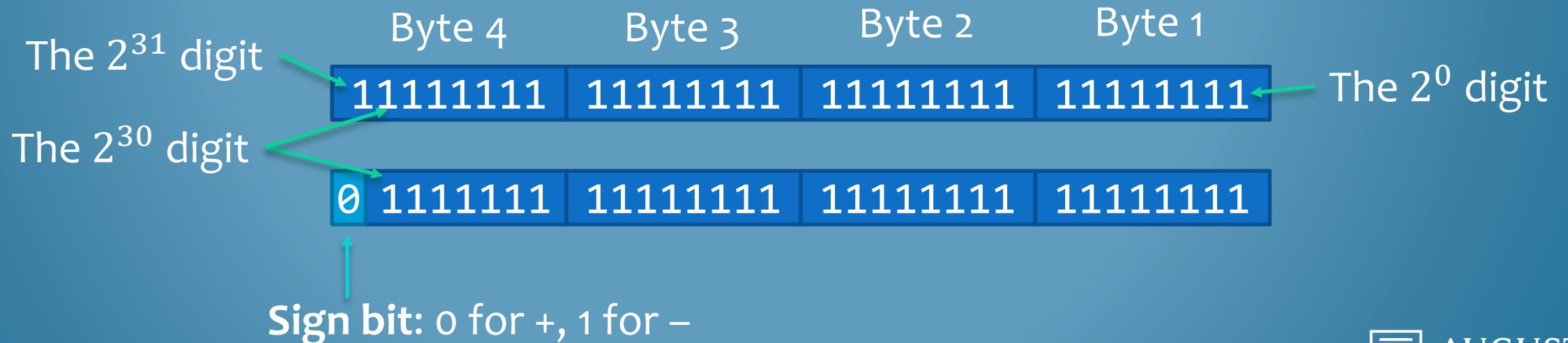
Integer Types by Size

- How much memory do you use to store each variable?
 - Depends on the size of the number
- `int` = 4 bytes = $-2^{31} \dots 2^{31} - 1$
- `long` = 8 bytes = $-2^{63} \dots 2^{63} - 1$
- `short` = 2 bytes = $-2^{15} \dots 2^{15} - 1$
- `byte` = 1 byte = $0 \dots 255$
 - `byte` is *unsigned* by default



Signed vs. Unsigned

- Unsigned versions of types can store larger (positive) values
 - `ushort` = 2 bytes = $0 \dots 2^{16} - 1$
 - `uint` = 4 bytes = $0 \dots 2^{32} - 1$
 - `ulong` = 8 bytes = $0 \dots 2^{64} - 1$
- Why? Storing the plus/minus sign takes one bit



Outline

- More About C# Datatypes
 - Integer types and sizes
 - **Floating-point type precision**
 - Value vs. Reference Types

Floating-Point Types by Size

- How many significant figures do you care about?

Type	Size	Range of Values	Digits of Precision
float	4 bytes	$\pm 1.5 \cdot 10^{-45} \dots \pm 3.4 \cdot 10^{38}$	7
double	8 bytes	$\pm 5.0 \cdot 10^{-324} \dots \pm 1.7 \cdot 10^{308}$	15-16
decimal	16 bytes	$\pm 1.0 \cdot 10^{-28} \dots \pm 7.9 \cdot 10^{28}$	28-29

- `float` and `double`: *approximations* of decimal values
- `decimal`: *exact* decimal values, as long as they fit in range

Approximate Decimals

- float and double use **binary** (base 2) fractions:

2^2 2^1 2^0 2^{-1} 2^{-2}
 101.01
 $= 4 + 1 + \frac{1}{4} = 5.25$

- Why “floating point”? Actually it’s binary scientific notation:

Mantissa \rightarrow **10101** \leftarrow Exponent **e-10**

Shift “.” left by 2 \rightarrow **101.01**

Or read it as a decimal \rightarrow $21 \cdot 2^{-2} = \frac{21}{4} = 5.25$

Approximate Decimals

- Some decimal (base 10) fractions can't be expressed in binary
- Decimal $0.1 = \frac{1}{10}$. Can you express $\frac{1}{10}$ as a sum of $\frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \dots$?

$$\begin{array}{ccccccc} \frac{1}{16} & \frac{1}{32} & \frac{1}{256} & \frac{1}{512} & & & \\ \downarrow & \downarrow & \downarrow & \downarrow & & & \\ .000110011\dots & & & & & & \end{array} \quad = .099609375$$

$(\dots 00110011\dots)$

- Truncating the infinite sequence leads to errors, just like with $\frac{1}{3}$:

$$\frac{1}{3} + \frac{2}{3} \text{ in decimal} = .33333333 + .66666666 = .99999999 \neq 1$$

Exact Decimals

- The `decimal` type uses decimal (base 10) fractions
- Finite decimals stay finite, so no unexpected errors
- But computers can only use binary, how does that work?

Decimal 5.1:

Mantissa → **110011** e_{10}^{-1} ← Exponent, but as a power of 10

↓ ↓

51 **10^{-1}**

$$51 \cdot 10^{-1} = \frac{51}{10} = 5.1$$

Choosing a Floating-Point Type

- More precision sounds good, why not always use decimal?

Type	Size	Range of Values	Digits of Precision
float	4 bytes	$\pm 1.5 \cdot 10^{-45} \dots \pm 3.4 \cdot 10^{38}$	7
double	8 bytes	$\pm 5.0 \cdot 10^{-324} \dots \pm 1.7 \cdot 10^{308}$	15-16
decimal	16 bytes	$\pm 1.0 \cdot 10^{-28} \dots \pm 7.9 \cdot 10^{28}$	28-29

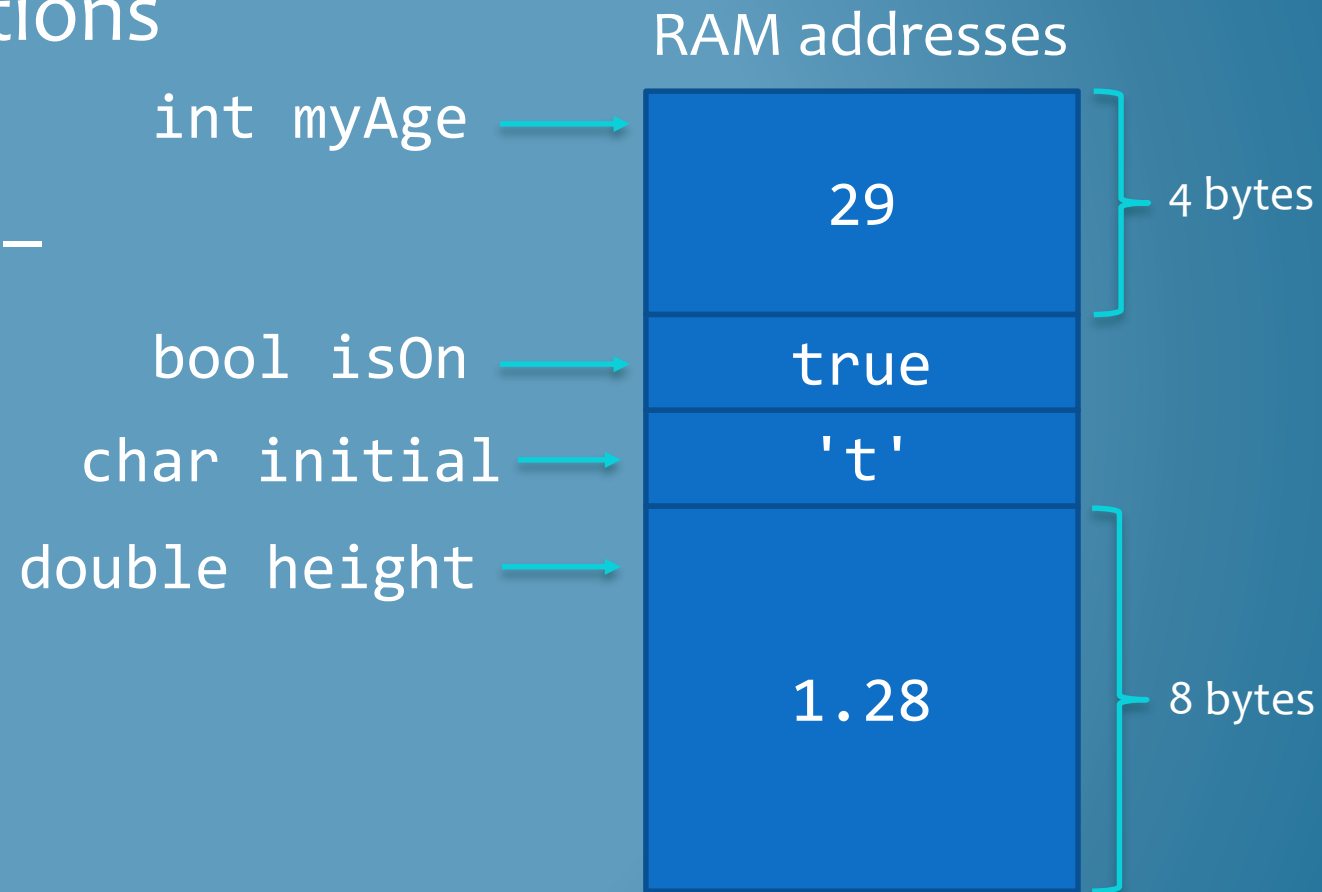
- **Much** slower to compute with than double
 - Base-10 exponent is hard, base-2 exponent is easy
- Only use if you really need exact values of .01, i.e. money
 - Physical measurements are not that exact

Outline

- More About C# Datatypes
 - Integer types and sizes
 - Floating-point type precision
 - **Value vs. Reference Types**

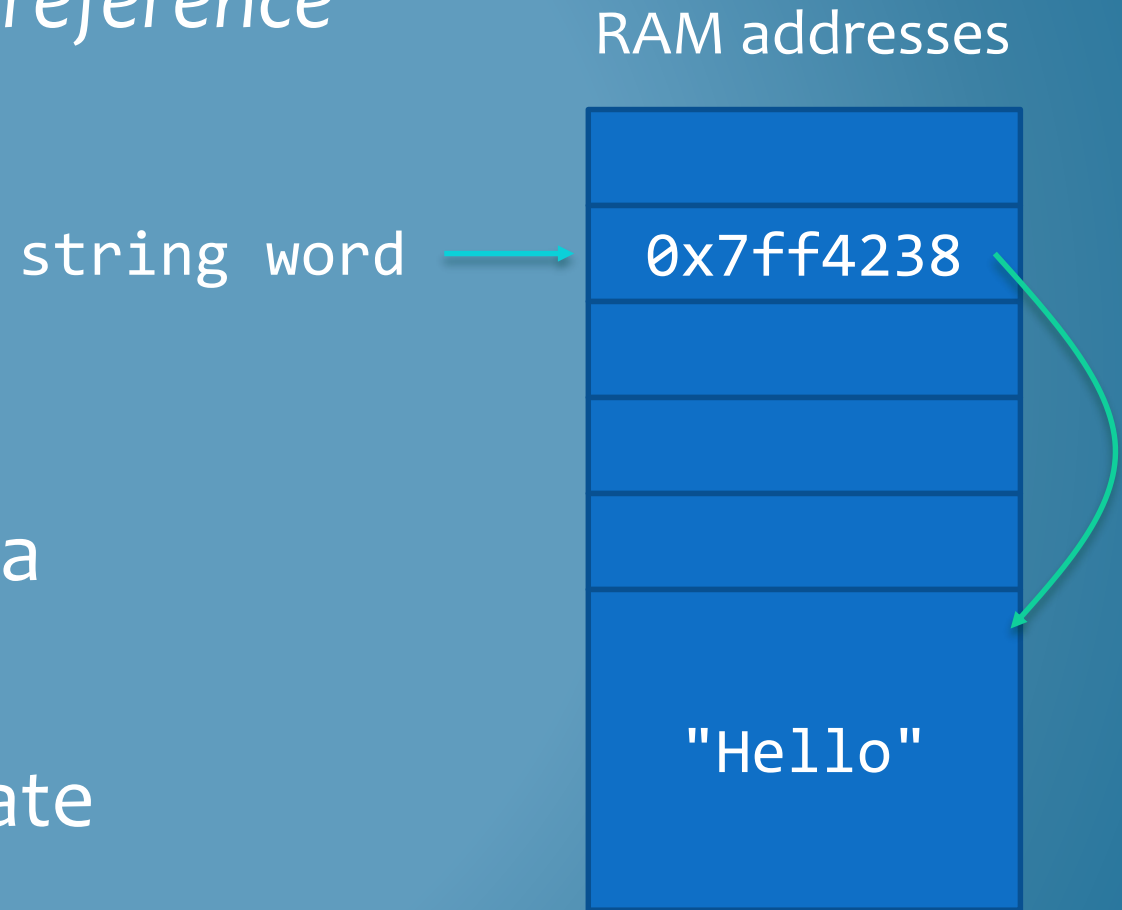
Value Types

- Variables name memory locations
- **Value Type:** Named memory location just stores the value – what you'd expect
- Numeric types (`int`, `uint`, `double`, `decimal`, etc.)
- `char`
- `bool` – logical true or false



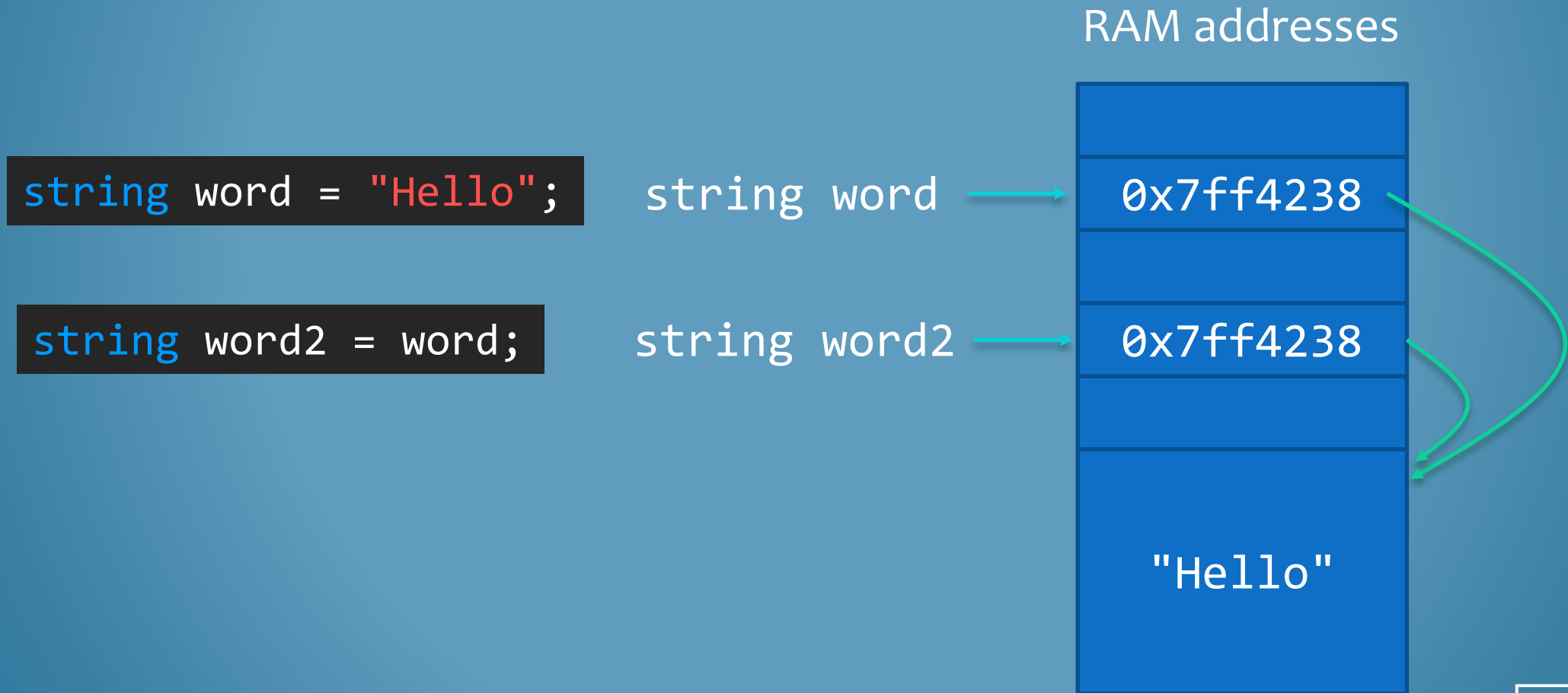
Reference Types

- Named memory location stores a *reference* to the data, not the data itself
- Contents of the named memory location = the address of another memory location
- This location stores the actual data
- `string`
- `object` – and all objects you create



Implications of Reference Types

- Assignment copies the value, but **not** the data it refers to



Conclusion

- More About C# Datatypes
 - Integer types and sizes
 - Floating-point type precision
 - Value vs. Reference Types