

Review

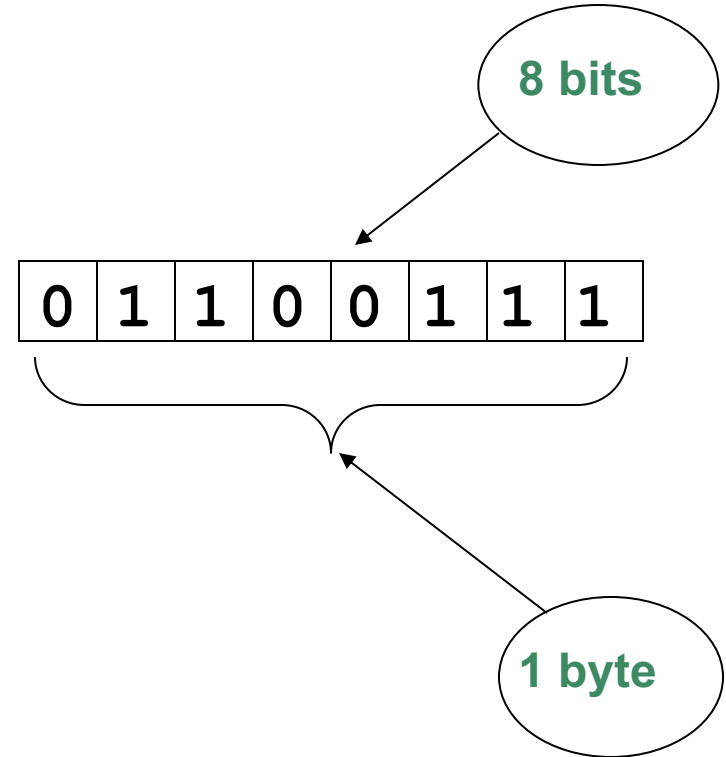
- For example, storing the number 4 (an integer) is a two-step process:
 - First, *allocate memory* by declaring a variable of type integer

```
int my_variable;
```
 - Then, *assign* it the piece of data

```
My_variable = 4;
```
 - We say that `my_variable` has the *value* 4

Main Memory Organization

- **Bit**
 - **Smallest** piece of memory
 - Stands for binary digit
 - Has values 0 (off) or 1 (on)
- **Byte**
 - Is 8 consecutive bits
- **Integer**
 - Usually 4 consecutive bytes
 - Has an address
 - 4 bytes = 32 bits



Data Storage

- All data is stored in binary bits in main memory
- The meaning of a set of bits depends on the *encoding*
 - Encoding determines the length of a meaningful chunk
 - We use bytes (8-bits) as our most common unit
 - Encoding also determines how those bits should be read

8-bit Binary	8-bit Integer	8-bit Character (ASCII)
0100 0001	65	'A' (1 byte = 8 bites)
0100 0010	66	'B'
0100 0011	67	'C'

Simple Data Types

- Data type: set of values together with a set of operations
- C++ data types fall into three categories:

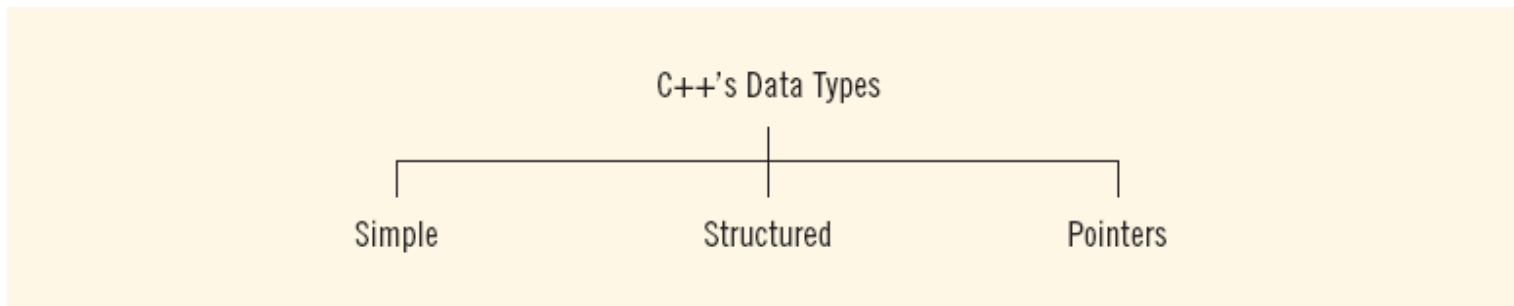


FIGURE 2-1 C++ data types

Simple Data Types

- Three categories of simple data
 - Integral: integers (numbers without a decimal)
 - Floating-point: decimal numbers
 - Enumeration type: user-defined data type

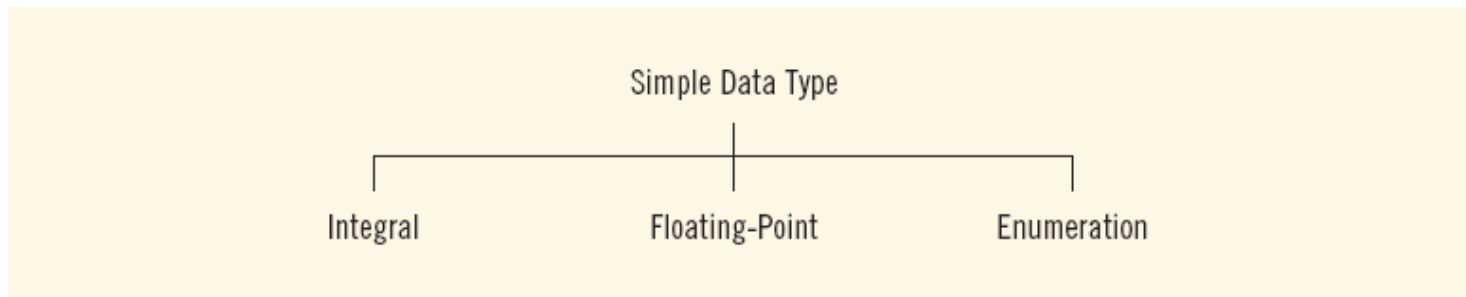
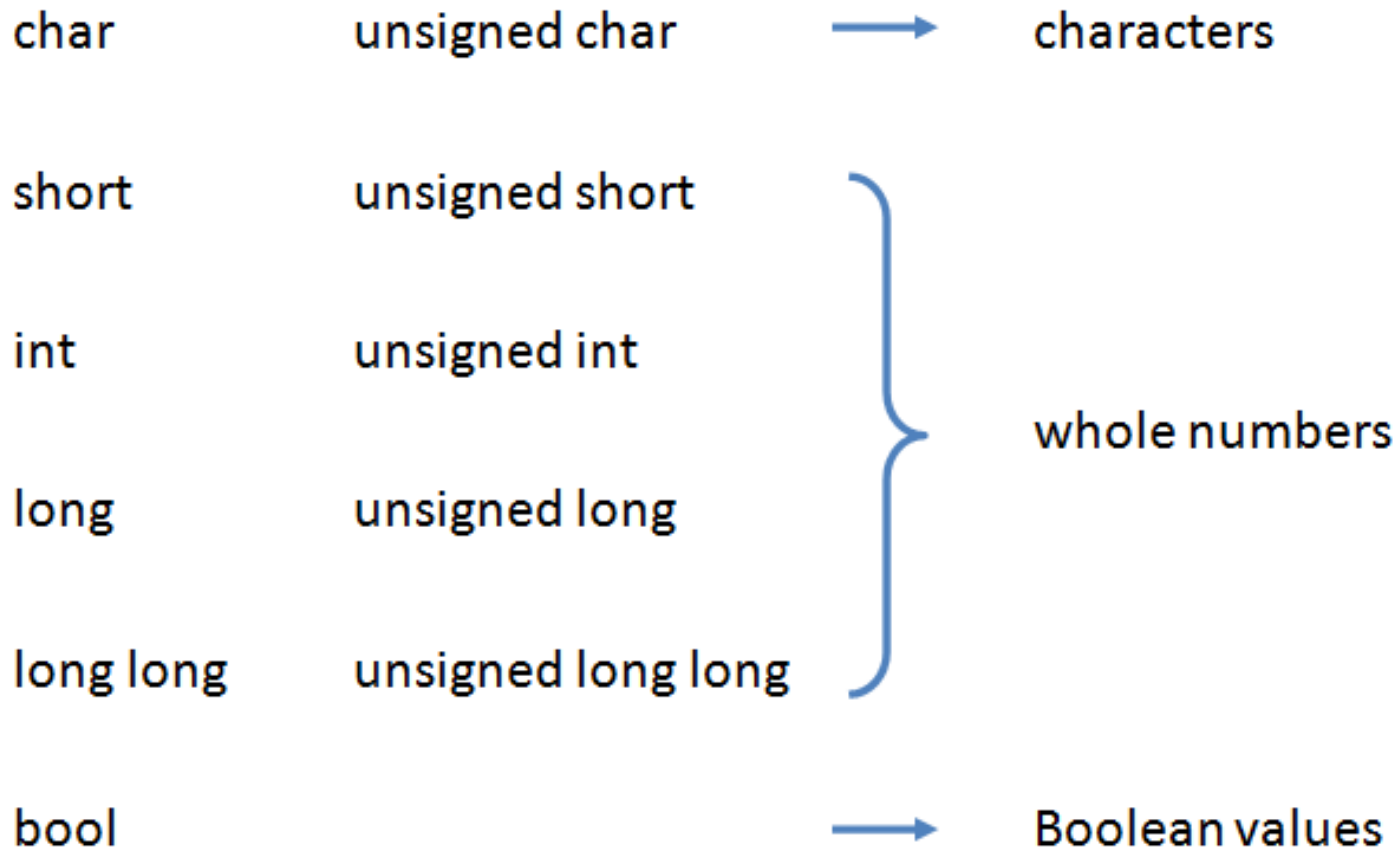


FIGURE 2-2 Simple data types

Simple Data Types (continued)

- Integral data types are further classified into the following categories:



int Data Type

- Examples:

– 6728

0

78

+763

- Positive integers do not need a + sign
- No commas are used within an integer
 - Commas are used for separating items in a list
 - Personal banker: 100,000

char Data Type

- The **smallest** integral data type
- Used for characters: letters, digits, and special symbols (**1 byte = 8 bites**)
- Each character is enclosed in single quotes
 - 'A', 'a', '0', '*', '+', '\$', '&', ' '
- A blank space is a character
 - char a;
 - a = ' ';

bool Data Type

- `bool` type
 - Two values: `true` and `false`
 - Manipulate logical (Boolean) expressions
- `true` and `false` are called **logical values**
- `bool`, `true`, and `false` are reserved words

Simple Data Types

TABLE 2-2 Values and Memory Allocation for Three Simple Data Types

Data Type	Values	Storage (in bytes)
<code>int</code>	-2147483648 to 2147483647	4 4*8=32 bits
<code>bool</code>	<code>true</code> and <code>false</code>	1 1 or 0 (one char): 8 bits
<code>char</code>	-128 to 127	1 8 bits

- Different compilers may allow different ranges of values

Floating-Point Data Types

- C++ uses **scientific notation** to represent real numbers (floating-point notation)

Scientific notation for math:

75.924 \Leftrightarrow 7.5924X10 \Leftrightarrow 7.5924XE1; 100 \Leftrightarrow E2; 0.1 \Leftrightarrow 10⁻¹ \Leftrightarrow E-1

TABLE 2-3 Examples of Real Numbers Printed in C++ Floating-Point Notation

Real Number	C++ Floating-Point Notation
75.924	7.592400E1
0.18	1.800000E-1
0.0000453	4.530000E-5
-1.482	-1.482000E0
7800.0	7.800000E3

Floating-Point Data Types (continued)

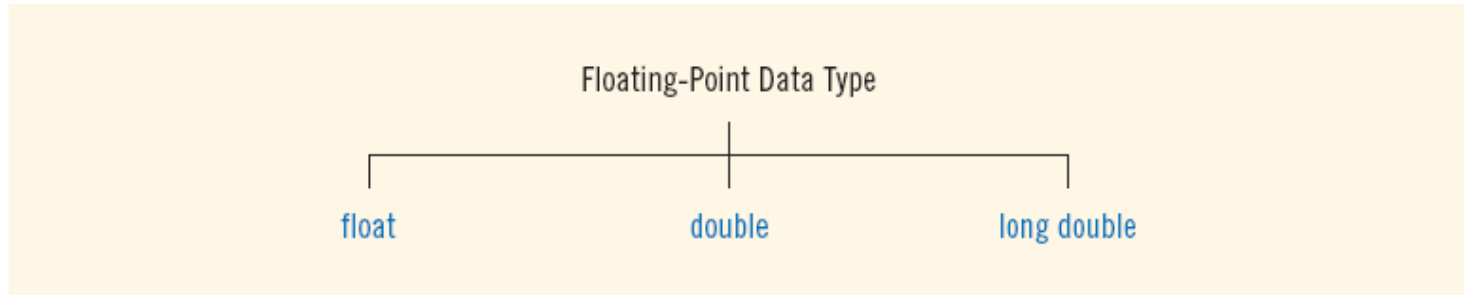


FIGURE 2-4 Floating-point data types

- `float`: represents any real number
 - Range: $-3.4\text{E}+38$ to $3.4\text{E}+38$ (four bytes)
- `double`: represents any real number
 - Range: $-1.7\text{E}+308$ to $1.7\text{E}+308$ (eight bytes)
- On most newer compilers, data types `double` and `long double` are same

Floating-Point Data Types (continued)

- Maximum number of significant digits (decimal places) for float values is 6 or 7
- Maximum number of significant digits for double is 15
- Precision: maximum number of significant digits
 - Float values are called single precision
 - Double values are called double precision

Arithmetic Operators

- C++ arithmetic operators:

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

- +, -, *, and / can be used with integral and floating-point data types
- % can be used with **integral** data types only
 - Not: 3.5 % 2
- These are *binary operators*
 - They operate on 2 *operands*

Order of Precedence

- All operations inside of $()$ are evaluated first
- $*$, $/$, and $\%$ are at the same level of precedence and are evaluated next
- $+$ and $-$ have the same level of precedence and are evaluated last
- When operators are on the same level
 - Performed **from left to right** (associativity)
- $3 * 7 - 6 + 2 * 5 / 4 + 6$ means
$$(((3 * 7) - 6) + ((2 * 5) / 4)) + 6$$

Overflow and Underflow

- Occurs when assigning a value that is **too large (overflow)** or **too small (underflow)** to be held in a variable
- Variable contains value that is 'wrapped around' the set of possible values

Overflow Example

- `// Create a short int initialized to`
- `// the largest value it can hold`
- `// [-32767, 32767]`
- `short int num = 32767;`

- `cout << num; // Displays 32767`
- `num = num + 1;`
- `cout << num; // Displays -32768`

Handling Overflow and Underflow

- Different systems handle the problem differently. They may
 - display a warning / error message
 - display a dialog box and ask what to do
 - stop the program
 - continue execution with the incorrect value

Expressions

Entire expression is evaluated according to precedence rules

- If **all** operands are integers
 - Expression is called an integral expression
 - Yields an integral result
 - Example: $2 + 3 * 5$
- If **all** operands are floating-point
 - Expression is called a floating-point expression
 - Yields a floating-point result
 - Example: $12.8 * 17.5 - 34.50$

Mixed Expressions

- Mixed expression:
 - Has operands of different data types
 - Contains integers and floating-point
- Examples of mixed expressions:

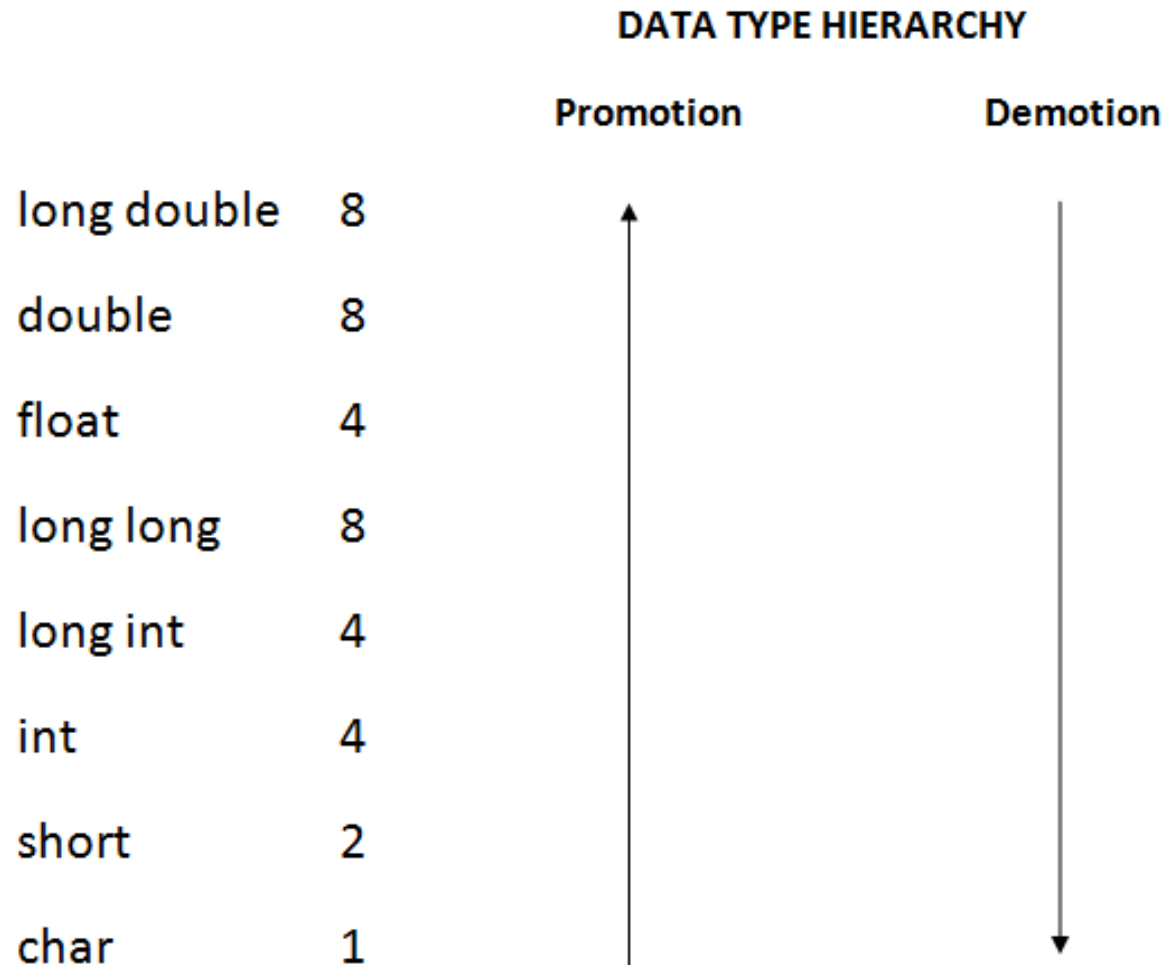
$$2 + 3.5$$

$$6 / 4 + 3.9 = 1 + 3.9 = 4.9$$

$$5.4 * 2 - 13.6 + 18 / 2$$

Mixed Expressions (cont'd.)

Data types are ranked by largest value they can hold



Mixed Expressions (cont'd.)

- Evaluation rules:
 - If expression has different types of operands
 - `char`, `short`, `unsigned short` are automatically promoted to `int`
 - All operands are promoted to the data type with the highest rank in the expression
 - When using the `=` operator, the type of the expression on `right` will be converted to type of the variable on `left` (promotion or demotion)

Data Types and Conversion

- The data type assigned to a literal or variable tells the computer how it is encoded
 - That is, how to interpret it
- You can also tell the computer to treat a literal or variable with a different encoding
 - This is called *type casting* or *type coercion*

```
static_cast<dataTypeName>( expression )
```

Rounding off a number

Suppose we need to round a real number to the nearest hundredth (2 decimal digits), we could do it by using the following procedure:

Number to be rounded off: **2.3449**

- 1) Move decimal point 2 places to the right -> $2.3449 * 100.0 \Rightarrow 234.49$
- 2) Add 0.5 -> $234.49 + 0.5 \Rightarrow 234.99$
- 3) Round the result downward (get the largest integral value that is not greater than the result -> $\Rightarrow 234.0$
- 4) Divide it by 100.0 -> $234.0 / 100.0 \Rightarrow$ **2.34**

Suppose we have a value in a variable called length (type double) and we want to round it off, we could use the following expression to implement the steps shown above:

$$\text{length} = \text{floor}((\text{length} * 100.0) + 0.5) / 100.0$$