SENG 1110 – Fall 2025
Chapter 2: Intro to C++ Programming

Dr. Zeinab Teimoori
Department of Engineering

THOMPSON RIVERS UNIVERSITY
Engineering

Pearson

Learning Objectives

- Write simple C++ applications.
- Use input and output statements.
- Use fundamental data types.
- Understand basic memory concepts.
- Use arithmetic operators.
- Understand the precedence of arithmetic operators.
- Write decision-making statements.
- Use relational and equality operators.

2.1 Comments

```
// fig02_01.cpp
// Text-printing program.
```

// is called a **single-line comment**

```
/* fig02_01.cpp: Text-printing program. */

/* fig02_01.cpp
   Text-printing program. */
```

multiline comments

```
#include <iostream> // enables program to output data to the screen
```

inline comments

2.2 Identifiers

- A variable name is any valid **identifier** that is not a keyword
- Consists of letters, digits and underscores (_)
- Must not begin with a digit
- C++ is **case sensitive**
  - a1 and A1 are different identifiers
- Avoid identifiers that begin with underscores and double underscores

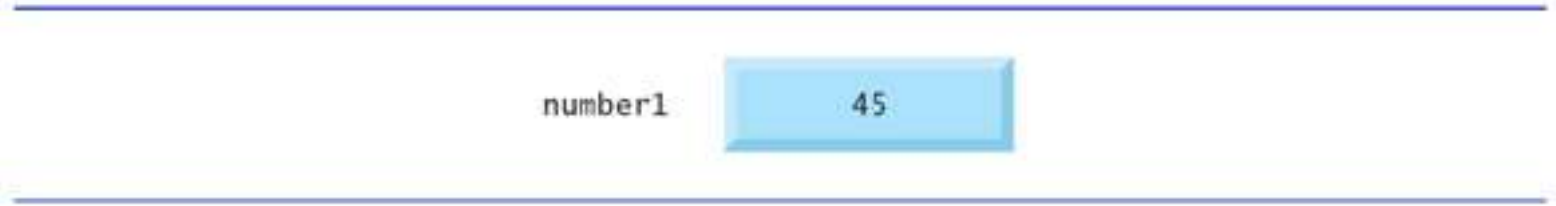
2.2 Variable Declarations and Braced Initialization (1 of 7)

- **Declarations** introduce identifiers into programs.
- The identifiers `number1`, `number2` and `sum` are the names of **variables**.
- A variable is a location in the computer's memory where a value can be stored for use by a program.
- Data of type **`int`**, means variables will hold **integers** (whole numbers such as 7, –11, 0 and 31914).

```
int number1 = 0;
int number2 = 0;
int sum = 0;
```

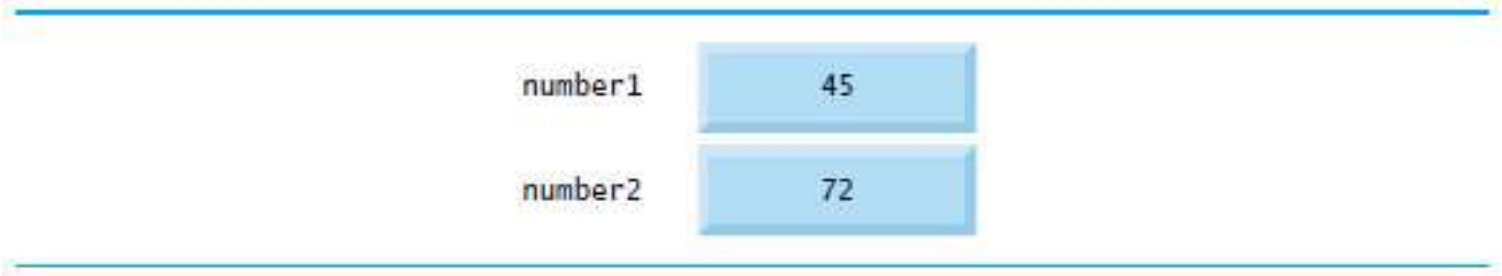
2.2 Variable Declarations and Braced Initialization (2 of 7)

Memory Location Showing the Name and Value of Variable `number1`



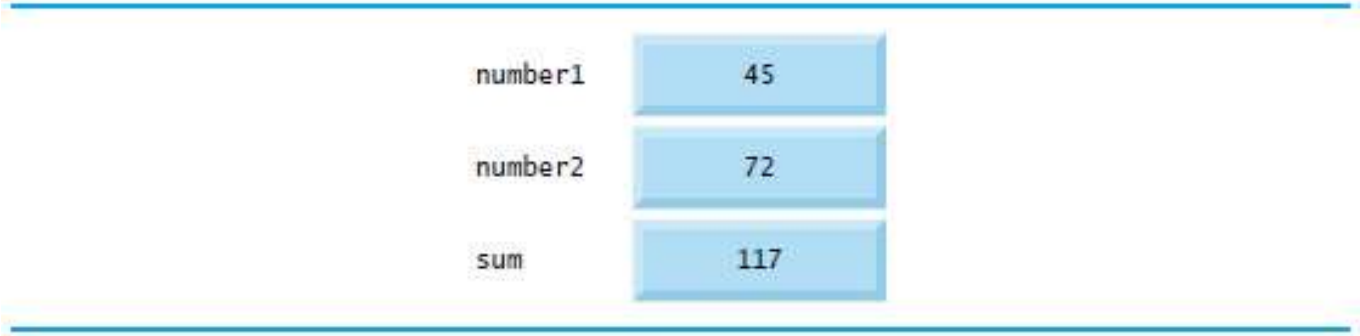
2.2 Variable Declarations and Braced Initialization (3 of 7)

Memory Locations After Storing Values in the Variables for `number1` and `number2`



2.2 Variable Declarations and Braced Initialization (4 of 7)

Memory Locations After Calculating and Storing the Sum of `number1` and `number2`



2.2 Variable Declarations and Braced Initialization (5 of 7)

- **int** number1{0}; *// first integer to add (initialized to 0)*
- **int** number2{0}; *// second integer to add (initialized to 0)*
- **int** sum{0}; *// sum of number1 and number2 (initialized to 0)*

- initialize each variable to 0 by placing a value in braces ({ and }) immediately following the variable's name
  - Known as list initialization

2.2 Variable Declarations and Braced Initialization (6 of 7)

- All variables **must** be declared with a **name** and a **data type** before they can be used in a program.

- **int** number1{0}, number2{0}, sum{0};

- If more than one name is declared in a declaration, the names are separated by commas (**,**);
- This is referred to as a **comma-separated list**.

2.2 Variable Declarations and Braced Initialization (7 of 7)

- Types such as `int`, `double` and `char` are called **fundamental types**.
- Data type `double` is for specifying real numbers, and data type `char` for specifying **character data**.
- Real numbers are numbers with decimal points, such as 3.4, 0.0 and –11.19.
- A `char` variable may hold only a single lowercase letter, a single uppercase letter, a single digit or a single special character (e.g., $ or *).
- Fundamental-type names are keywords and therefore **must** appear in all lowercase letters.

Checkpoint

(True/False): Together, blank lines, spaces and tabs are known as whitespace and are usually ignored by the compiler.

(True/False) The following statement displays the string "Hello" then positions the output cursor on the next line.

```
std::cout << "Hello";
```

2.3 Arithmetic Operators (1 of 4)

Operation	Arithmetic operator	C++ expression
Addition	+	f + 7
Subtraction	-	p - c
Multiplication	*	b * m
Division	/	x / y
Remainder	%	r % s

- **binary operators** because each has two operands.
- **Integer division** in which the numerator and the denominator are integers yields an integer quotient

17/5 evaluates to 3

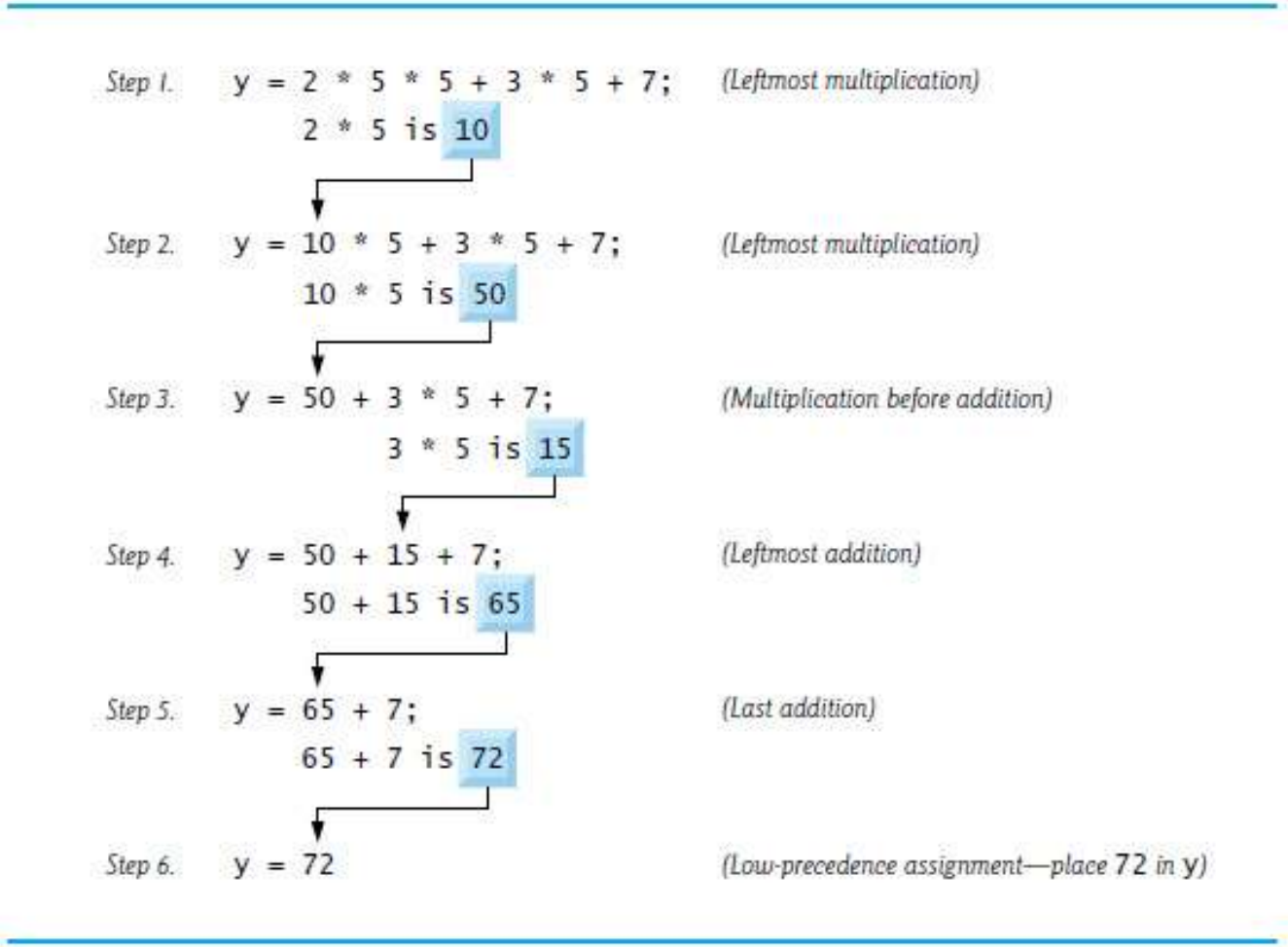
2.3 Arithmetic Operators (2 of 4)

- Parentheses are used in C++ expressions in the same manner as in algebraic expressions.
- For example, to multiply a times the quantity b + c we write
  - a * (b+c)
- **rules of operator precedence**

2.3 Arithmetic Operators (3 of 4)

Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. For **nested** parentheses, the expression in the **innermost** pair evaluates first.
* / %	Multiplication Division Remainder	Evaluated second. If there are several, they're evaluated left to right.
+ −	Addition Subtraction	Evaluated last. If there are several, they're evaluated left to right.

2.3 Arithmetic Operators (4 of 4)



Checkpoint

(Fill-in) Every variable has a name, a type and a _____.

(True/False) Storing a value in a variable replaces the previous value in the corresponding memory location.

2.4 Obtaining Value from the User (1 of 3)

- A **prompt** directs the user to take a specific action.
- A `cin` statement uses the **input stream object `cin`** (of namespace `std`) and the **stream extraction operator**, **`>>`**, to obtain a value from the keyboard.
- Using the stream extraction operator with `std::cin` takes character input from the standard input stream, which is usually the keyboard.

```
std::cin >> number1; // read first integer
from user into number1
```

2.4 Obtaining Value from the User (2 of 3)

- When the computer executes an input statement, it waits for the user to enter a value for variable `number1`.
- The user responds by typing the number (as characters) then pressing the **Enter** key (sometimes called the **Return** key) to send the characters to the computer.
- The computer converts the character representation of the number to an integer and assigns (i.e., copies) this number (or **value**) to the variable `number1`.
- Any subsequent references to `number1` in this program will use this same value.

2.4 Obtaining Value from the User (3 of 3)

```
1  // Fig. 2.5: fig02_05.cpp
2  // Addition program that displays the sum of two integers.
3  #include <iostream> // enables program to perform input and output
4
5  // function main begins program execution
6  int main() {
7     // declaring and initializing variables
8     int number1{0}; // first integer to add (initialized to 0)
9     int number2{0}; // second integer to add (initialized to 0)
10    int sum{0}; // sum of number1 and number2 (initialized to 0)
11
12    std::cout << "Enter first integer: "; // prompt user for data
13    std::cin >> number1; // read first integer from user into number1
14
15    std::cout << "Enter second integer: "; // prompt user for data
16    std::cin >> number2; // read second integer from user into number2
17
18    sum = number1 + number2; // add the numbers; store result in sum
19
20    std::cout << "Sum is " << sum << std::endl; // display sum; end line
21 } // end function main
```

2.5 Decision Making: Equality and Relational Operators (1 of 7)

- The **`if` statement** allows a program to take alternative action based on whether a **condition** is true or false.
  - If the condition is true, the statement in the body of the if statement is executed.
  - If the condition is false, the body statement is not executed.
- Conditions in `if` statements can be formed by using the **equality operators** and **relational operators.**
- The relational operators all have the same level of precedence and associate left to right.
- The equality operators both have the same level of precedence, which is lower than that of the relational operators, and associate left to right.

2.5 Decision Making: Equality and Relational Operators (2 of 7)

Relational operators

Algebraic relational or equality operator	C++ relational or equality operator	Sample C++ condition	Meaning of C++ condition
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

Equality operators

Algebraic relational or equality operator	C++ relational or equality operator	Sample C++ condition	Meaning of C++ condition
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y

2.5 Decision Making: Equality and Relational Operators (3 of 7)

```
1  // Fig. 2.13: fig02_13.cpp
2  // Comparing integers using if statements, relational operators
3  // and equality operators.
4  #include <iostream> // enables program to perform input and output
5
6  using std::cout; // program uses cout
7  using std::cin; // program uses cin
8  using std::endl; // program uses endl
9
10 // function main begins program execution
11 int main() {
12    int number1{0}; // first integer to compare (initialized to 0)
13    int number2{0}; // second integer to compare (initialized to 0)
14
15    cout << "Enter two integers to compare: "; // prompt user for data
16    cin >> number1 >> number2; // read two integers from user
17
18    if (number1 == number2) {
19       cout << number1 << " == " << number2 << endl;
20    }
21
```

2.5 Decision Making: Equality and Relational Operators (4 of 7)

```
22      if (number1 != number2) {
23         cout << number1 << " != " << number2 << endl;
24      }
25
26      if (number1 < number2) {
27         cout << number1 << " < " << number2 << endl;
28      }
29
30      if (number1 > number2) {
31         cout << number1 << " > " << number2 << endl;
32      }
33
34      if (number1 <= number2) {
35         cout << number1 << " <= " << number2 << endl;
36      }
37
38      if (number1 >= number2) {
39         cout << number1 << " >= " << number2 << endl;
40      }
41   } // end function main
```

2.5 Decision Making: Equality and Relational Operators (5 of 7)

```
Enter two integers to compare: 3 7
3 != 7
3 < 7
3 <= 7
```

```
Enter two integers to compare: 22 12
22 != 12
22 > 12
22 >= 12
```

```
Enter two integers to compare: 7 7
7 == 7
7 <= 7
7 >= 7
```

2.5 Decision Making: Equality and Relational Operators (6 of 7)

- Each `if` statement has a single statement in its body and each body statement is indented.
- Each `if` statement's body is enclosed in a pair of braces, `{ }`, creating what's called a **compound statement** or a **block** that may contain multiple statements.

2.5 Decision Making: Equality and Relational Operators (7 of 7)

Operators	Associativity	Type
()	**grouping**	grouping parentheses
* / %	left to right	multiplicative
+ -	left to right	additive
<< >>	left to right	stream insertion/extraction
< <= > >=	left to right	relational
== !=	left to right	equality
=	right to left	assignment

Checkpoint

(Code) Assume number1 is 12 and number2 is 23. What does the following code print?

```
if (number1 > number2) {
   cout << number1 << " > " << number2 << "\n";
}
```

Checkpoint

(Code) Assume number1 is 23 and number2 is 23. What does the following code print?

```
if (number1 != number2); {
   cout << number1 << " != " << number2 << "\n";
}
```