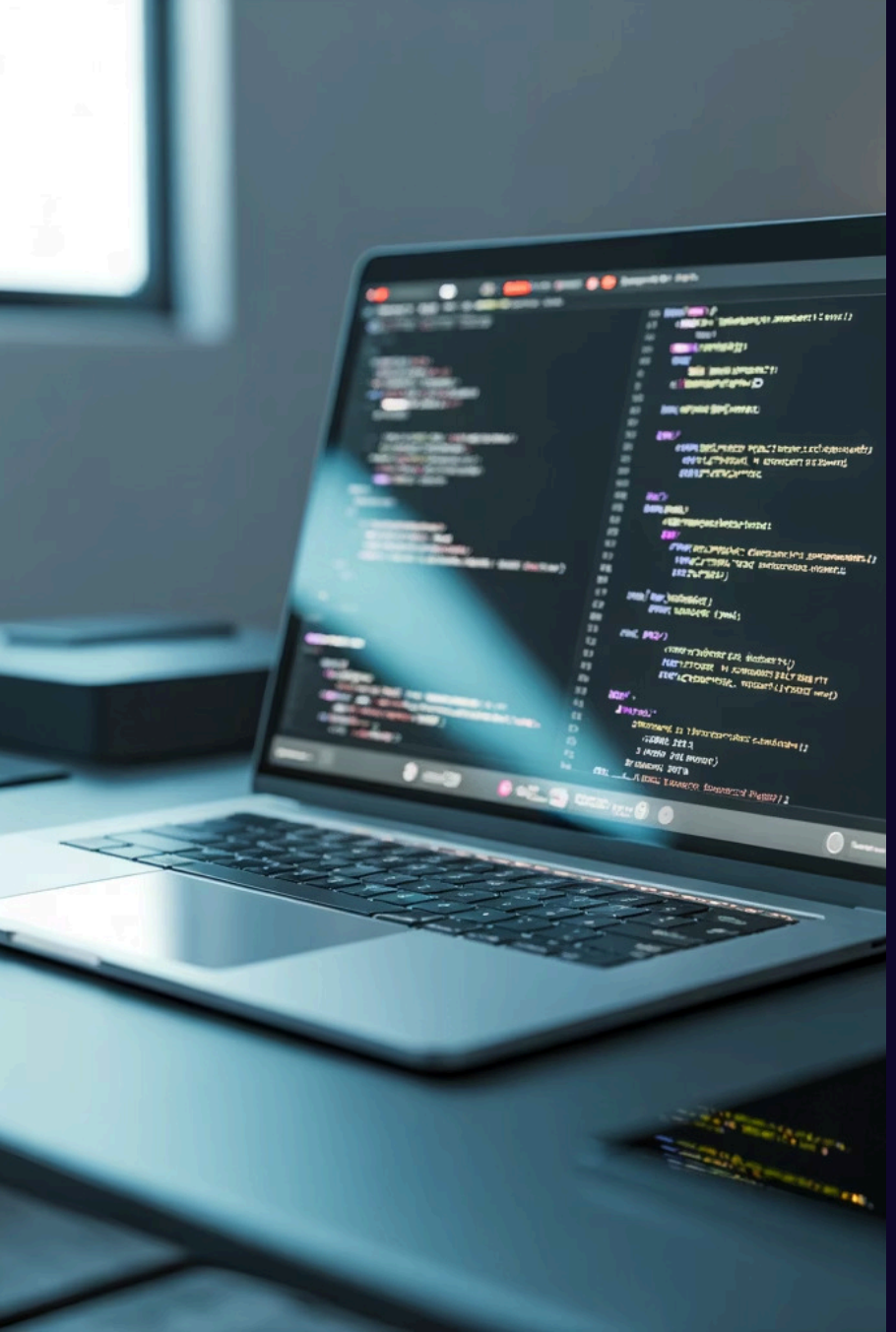

Programming Fundamentals with C++ — Week 3: cout, cin, Data Types, Variables & Arithmetic

This lecture covers the essentials beginners need to write simple input/output programs and do arithmetic in C++. We'll define each concept, show clear examples, and provide step-by-step explanations in plain English. Each slide includes real photos illustrating coding and learning environments to make ideas concrete.

What is cout? — Print output to the screen

Definition: **cout** is the standard C++ object used to send text and values to the console (the program's output window). It is part of the iostream library and works with the insertion operator <<.

Key points: - You must include `#include <iostream>` at the top of your program. - Use `std::cout` or a `using` statement to avoid typing `std::` each time. - You can print text, variables, and expressions.

Example code:

```
#include <iostream>
using namespace std;

int main() {
  cout << "Hello, world!" << endl;
  int x = 5;
  cout << "x = " << x << endl;
  return 0;
}
```

Explanation: `endl` moves the cursor to the next line and flushes the output. The insertion operator << chains multiple items to print in sequence.

What is cin? — Read input from the user

Definition: **cin** is the standard C++ object used to read input from the keyboard (standard input). It uses the extraction operator >> to put values into variables.

Key points: - Include `#include <iostream>`. - Input stops at whitespace (space, tab, newline) for basic types. - Always read into a variable whose type matches the expected input.

Example code:

```
#include <iostream>
using namespace std;

int main() {
  int age;
  cout << "Enter your age: ";
  cin >> age;
  cout << "You entered: " << age << endl;
  return 0;
}
```

Explanation: The program pauses at `cin >> age;` waiting for the user to type a number and press Enter. If input type doesn't match, cin enters a fail state — later slides show how to check and recover.

Basic Data Types — integer, float, double, char

Definitions and use-cases: - **int**: whole numbers (no decimals). Use when counting or indexing. Typical range depends on system. - **float**: single-precision decimal numbers. Good for simple fractions but limited precision. - **double**: double-precision decimals. Use this for more accurate calculations. - **char**: single character (like 'A' or 'x'). Stored as a small integer internally.

Example code showing all four types:

```
#include <iostream>
using namespace std;

int main() {
  int count = 10;
  float temp = 98.6f;
  double pi = 3.1415926535;
  char initial = 'J';

  cout << "count: " << count << endl;
  cout << "temp: " << temp << endl;
  cout << "pi: " << pi << endl;
  cout << "initial: " << initial << endl;
  return 0;
}
```

Notes: Use `f` after a float literal (98.6f) to tell the compiler it is a float. Prefer `double` for most math unless memory is critical.

Variables — storing information

Definition: A **variable** is a named place in memory where you store a value. You declare a variable by specifying its type and name, and you can assign a value to it.

Rules and best practices: - Choose descriptive names (e.g., `totalScore`, not `t`). - Follow naming rules: letters, digits, and underscores; cannot start with a digit. - Initialize variables when you declare them to avoid undefined values.

Example code:

```
#include <iostream>
using namespace std;

int main() {
  int totalScore = 0;      // start at zero
  double average = 0.0;    // decimals allowed
  char grade = 'A';        // single character

  cout << "Score: " << totalScore << ", Grade: " << grade << endl;
  return 0;
}
```

Explanation: Variables make your program flexible. You can change values at runtime through calculations or user input.

Arithmetic Operators: +, -, *, /

Definition: Arithmetic operators let you perform basic math on numeric variables and literals. - **+** addition - **-** subtraction - ***** multiplication - **/** division

Important details: - Integer division drops the fractional part (e.g., 7 / 2 == 3). - For decimals use `float` or `double` (7.0 / 2.0 == 3.5). - Operator precedence: * and / run before + and -. Use parentheses to control order.

Example code with mixed types:

```
#include <iostream>
using namespace std;

int main() {
  int a = 7, b = 2;
  cout << "a + b = " << (a + b) << endl;
  cout << "a - b = " << (a - b) << endl;
  cout << "a * b = " << (a * b) << endl;
  cout << "a / b (int) = " << (a / b) << endl;

  double da = 7.0, db = 2.0;
  cout << "da / db (double) = " << (da / db) << endl;
  return 0;
}
```

Tip: When mixing ints and doubles, C++ converts ints to doubles for the calculation (called type promotion).

Arithmetic Expressions — combining operators and variables

Definition: An **arithmetic expression** is any combination of variables, literals, and operators that evaluates to a single value. Expressions can include parentheses to group operations and function calls for more complex math.

Common patterns: - Assign results to variables: `result = (a + b) * c;` - Update variables in place: `count += 1;` is the same as `count = count + 1;` - Use parentheses to avoid ambiguity and make code readable.

Example: compute average and area:

```
#include <iostream>
using namespace std;

int main() {
  double x = 10.0, y = 20.0, z = 30.0;
  double average = (x + y + z) / 3.0;
  double radius = 5.0;
  double area = 3.1415926535 * radius * radius;

  cout << "Average: " << average << endl;
  cout << "Circle area: " << area << endl;
  return 0;
}
```

Debugging tip: Print intermediate values with cout to verify each part of the expression if results look wrong.

Putting It All Together — interactive calculator example

This example uses cin, cout, multiple data types, variables, and arithmetic to build a small interactive calculator. It shows clear input prompts, error avoidance by using doubles, and simple output formatting.

```
#include <iostream>
using namespace std;

int main() {
  double num1, num2;
  char op;

  cout << "Enter first number: ";
  cin >> num1;
  cout << "Enter an operator (+ - * /): ";
  cin >> op;
  cout << "Enter second number: ";
  cin >> num2;

  double result;
  if (op == '+') result = num1 + num2;
  else if (op == '-') result = num1 - num2;
  else if (op == '*') result = num1 * num2;
  else if (op == '/') {
    if (num2 == 0.0) {
      cout << "Error: Division by zero" << endl;
      return 1;
    }
    result = num1 / num2;
  } else {
    cout << "Unknown operator" << endl;
    return 1;
  }

  cout << "Result: " << result << endl;
  return 0;
}
```

Learning notes: Use doubles to support fractional input. Check for division by zero to prevent crashes or incorrect behavior.

Common Beginner Pitfalls & Best Practices

1

1. Uninitialized variables

Always initialize variables. An uninitialized int may contain garbage and lead to incorrect results.

2

2. Integer division surprises

7 / 2 using ints gives 3. Use 7.0 / 2.0 or cast to double for decimals.

3

3. Type mismatch on input

If cin expects an int and user types text, cin fails. Validate input or read into strings and convert carefully.

4

4. Remember header & namespace

Include <iostream> and either use std:: or `using namespace std;` for simpler code while learning.

Quick fixes: Initialize variables at declaration, use doubles for calculations needing fractions, and add simple input checks after cin (e.g., `if (!cin)`).

Summary, Practice Exercises & Next Steps

Summary: - cout prints output; cin reads input. - Basic types: int, float, double, char — choose the right type for your data. - Variables store information and must be declared with a type. - Arithmetic operators (+, -, *, /) let you compute values; be careful with integer division and operator order.

Practice exercises (try these on your own): 1) Write a program that asks for two integers and prints their sum, difference, product, and quotient (show decimal quotient). 2) Ask for a user's first initial (char) and age (int), then print a greeting that includes both. 3) Create a program to compute the area and circumference of a circle from a radius input (use double).

Next steps: In Week 4 we'll cover control flow (if statements and loops), which lets your programs make decisions and repeat actions. Before then, try the exercises above and re-run the calculator example while changing inputs to understand how types and operators behave.