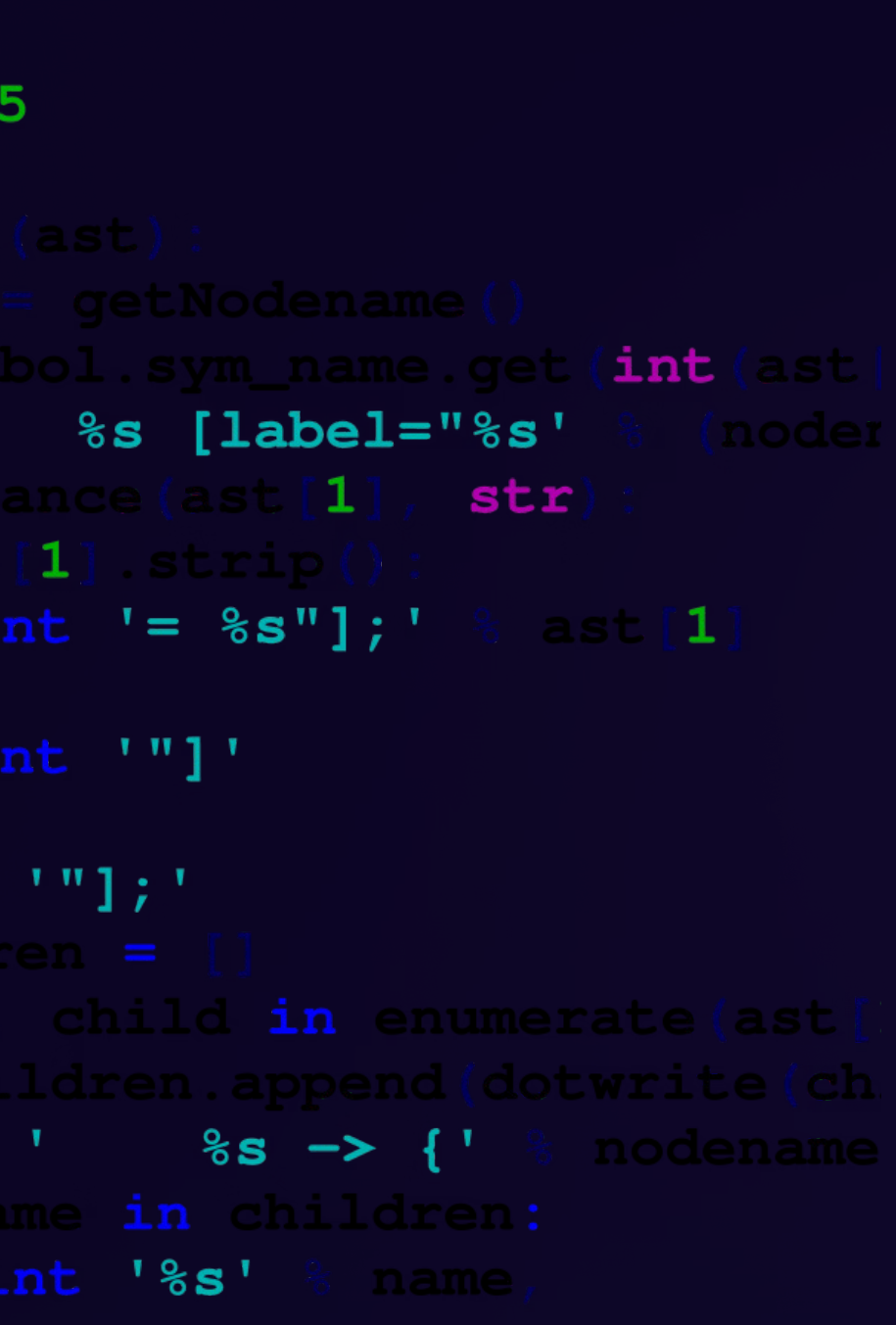

Programming Fundamentals with C++

Week 1: Variables, Memory, and Basic Operations

Welcome to your journey into C++ programming! This course will teach you the essential building blocks of computer programming, starting with how computers store and manage information. By the end of this week, you'll understand variables, memory, basic data types, and how to work with user input and arithmetic operations.

What is a Variable? Understanding the Basics

A variable is like a named box in your computer's memory. Just as you might label a box in a storage unit to remember what's inside, a variable has a name and holds information (called a value). When your program runs, this box exists somewhere in your computer's RAM (Random Access Memory).

Think about it this way: if you're writing a program to track student grades, you might create a variable named `studentScore` to store a number like 85 or 92. Every time your program needs that score, it simply looks it up by the variable's name instead of remembering the exact memory address.

Variable Name

A label you choose (like `age` or `temperature`)

Data Type

What kind of data it holds (number, decimal, text, true/false)

Value

The actual information stored inside

Memory Concepts: How Computers Store Your Data

Your computer's memory is organized like an apartment building with numbered units. Each unit can hold information, and each unit has a unique address. When you create a variable, the computer reserves one or more units to store your data. The variable name is just a friendly way to refer to that address.

RAM (Random Access Memory)

This is temporary, lightning-fast memory that your program uses while running. When you turn off your computer, everything in RAM disappears—it's erased. Think of it as your computer's short-term thinking space.

Memory Address

Each location in memory has a unique number (like 0x7fff5fbff8c0). You don't usually need to know these numbers directly because the variable name handles it for you. But knowing they exist helps you understand what's happening behind the scenes.

Why does this matter? Understanding memory helps you write efficient programs and avoid bugs. For example, when you declare a variable, the computer figures out how much memory it needs based on the data type. An integer uses 4 bytes, while a decimal number (called a float) also uses 4 bytes in C++.

Integer Variables: Working with Whole Numbers

An integer variable stores whole numbers—positive, negative, or zero. In C++, we use the keyword `int` to declare an integer. Integers are the most common data type in programming because they're simple, fast, and perfect for counting, storing ages, scores, quantities, and many other things.

Understanding Integer Ranges

A standard integer in C++ can hold numbers from approximately **-2.1 billion to 2.1 billion**. This range exists because an integer uses 4 bytes (32 bits) of memory, and each bit can represent either 0 or 1. If you need bigger numbers, C++ offers `long long`, which uses 8 bytes.

Declaration Example

```
int age;
int score = 95;
int temperature = -5;
```

Real-World Uses

- Student grades (0-100)
- Number of items in inventory
- Player position in a game
- Year (2024)

Notice in the examples above: `int age;` declares a variable but doesn't set a value yet (it holds garbage data), while `int score = 95;` declares and immediately assigns the value 95. The third line shows negative numbers work too.

Floating Point Variables: Numbers with Decimals

Floating point variables store numbers with decimal points. In C++, we use the keyword `float` for regular decimal numbers or `double` for more precise decimal numbers. "Floating point" refers to the decimal point's position, which can "float" to different places depending on the magnitude of the number.

Float vs Double: What's the Difference?

Float

Uses 4 bytes. Good for most everyday needs. Can store roughly 7 significant decimal digits accurately. Example: 3.14159

Double

Uses 8 bytes. Provides about 15 significant decimal digits. Better for scientific calculations and when precision matters. Example: 3.14159265358979

Float Declaration Example

```
float price = 19.99f;
float gpa = 3.85f;
float height = 5.9f;
```

Double Declaration Example

```
double pi = 3.14159265;
double distance = 1234.567891;
double temperature = 98.6;
```

Pro Tip: Notice the `f` after float values (like `19.99f`)? This tells C++ "treat this as a float." Without it, C++ assumes it's a double, which can cause issues. For doubles, you don't need any special character.

Initialization: Setting Up Your Variables Correctly

Initialization means giving a variable its first value when you create it. This is crucial because uninitialized variables contain random "garbage" data that can cause unpredictable behavior in your program. As a beginner programmer, always initialize your variables!

Three Ways to Initialize Variables in C++

1 Copy Initialization

This is the traditional way: `int age = 25;` The equals sign copies the value on the right into the variable on the left. It's simple and readable.

2 Direct Initialization

Using parentheses: `int age(25);` This is less common in basic C++ but means the same thing. You're directly constructing the variable with a value.

3 Uniform Initialization

Using curly braces: `int age{25};` This is modern C++ (C++11 and newer) and is considered the safest method because it prevents accidental type conversions that could cause data loss.

Best Practice: As a beginner, use copy initialization `int age = 25;` since it's the clearest and most widely used. As you advance, learn about uniform initialization for safety.

Taking Input from Users with cin

`cin` (pronounced "see-in") is C++'s way of letting your program read information that users type on their keyboard. It stands for "character input" and is part of the standard library. Without `cin`, your programs could only use data you hardcode into them. With `cin`, your programs become interactive and responsive to user needs.

How cin Works

When your program encounters a `cin` statement, it pauses and waits for the user to type something. Once the user presses Enter, the typed information flows into your variable like water pouring into a cup. The operator `>>` (two greater-than signs) is called the "extraction operator"—it extracts data from the keyboard and puts it into your variable.

Basic cin Syntax

```
cin >> variableName;
```

This waits for user input and stores it in the variable.

Complete Example

```
int studentAge;
cout << "Enter your age: ";
cin >> studentAge;
cout << "You are " << studentAge
    << " years old!";
```

Notice: We use `cout << "..."` to display a question, then `cin >> studentAge` to receive the answer. Always prompt the user before asking for input, so they know what to type!

Arithmetic Operators: Basic Math in C++

Arithmetic operators perform mathematical calculations: addition, subtraction, multiplication, and division. C++ uses familiar symbols for these operations, making it natural to write math in your code. These operators work with both integer and floating-point variables, though the results can differ subtly between the two.

The Four Basic Arithmetic Operators

Addition (+)

`5 + 3 = 8`

Adds two numbers together.

Subtraction (-)

`10 - 4 = 6`

Subtracts the second number from the first.

Multiplication (*)

`6 * 7 = 42`

Multiplies two numbers together.

Division (/)

`20 / 4 = 5`

Divides first number by the second.

Important Note About Division: When you divide two integers in C++, the result is truncated (the decimal part is thrown away). So `7 / 2` equals `3`, not `3.5`. To get the decimal result, at least one number must be a floating-point type.

Arithmetic Expressions: Combining Operations and Understanding Order

An arithmetic expression combines variables, numbers, and operators into a single calculation. C++ follows a standard order of operations (called precedence) just like standard mathematics: multiplication and division happen before addition and subtraction, and operations of equal precedence happen left to right. Understanding this prevents mistakes in your calculations.

Order of Operations (PEMDAS in C++)

1. **Parentheses ()** – Highest priority
2. **Multiplication (*) and Division (/)** – Equal priority, left to right
3. **Addition (+) and Subtraction (-)** – Lowest priority, left to right

Order of Operations Examples

```
int result1 = 2 + 3 * 4;    // = 14 (not 20)
// Multiply first: 3 * 4 = 12
// Then add: 2 + 12 = 14

int result2 = (2 + 3) * 4;   // = 20
// Parentheses first: 2 + 3 = 5
// Then multiply: 5 * 4 = 20

int result3 = 20 / 4 / 2;    // = 2.5
// Left to right: 20/4 = 5
// Then: 5 / 2 = 2
```

Real-World Expression

```
double totalPrice =
    (itemPrice * quantity) + tax;

int average = (score1 + score2 + score3)
    / 3;

double kinetic_energy =
    0.5 * mass * velocity * velocity;
```

When in doubt, use parentheses! They make your code clearer and prevent unexpected results. `(2 + 3) * 4` is much easier to understand than relying on people remembering the order of operations.

Putting It All Together: A Complete Program Example

Let's create a complete, working C++ program that uses everything we've learned: variable declaration, initialization, user input with `cin`, and arithmetic operations. This program calculates the area of a rectangle when the user provides the dimensions.

Complete Working Code

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

int main() {
  double length, width, area;

  cout << "Welcome to Rectangle Calculator!\n";
  cout << "Enter length: ";
  cin >> length;

  cout << "Enter width: ";
  cin >> width;

  area = length * width;

  cout << "Area = " << area
       << " square units\n";

  return 0;
}
```

What This Program Does

- **Lines 1-2:** Include input/output library
- **Lines 5-6:** Declare three floating-point variables
- **Lines 8-11:** Display welcome and get length
- **Lines 13-14:** Get width from user
- **Line 16:** Calculate area using multiplication
- **Lines 18-19:** Display the result
- **Line 21:** End the program

Sample Program Execution

```
Welcome to Rectangle Calculator!
Enter length: 5.5
Enter width: 3.2
Area = 17.6 square units
```

Congratulations! You now understand the fundamental concepts of C++ programming. Practice by creating similar programs: calculate the area of a circle, convert temperatures, calculate simple interest, or find an average grade. Each program will strengthen your understanding of variables, input, and arithmetic operations.