C++ Zero to Hero — A Beginner's Guide

A free guide by Parth Shah — ajconsultation.com
One-on-one tutoring in St. Augustine, Florida, or online.

Overview

This guide walks you through C++ from the ground up over 8 weeks. It is designed for students in their first college CS course, those interested in competitive programming, or anyone who wants to understand how computers actually manage memory.

C++ is harder to learn than Python or Java. The syntax is stricter, there is less handholding, and a mistake with memory can crash your program in confusing ways. But that is exactly the point. Students who work through C++ come out the other side with a much deeper understanding of what a computer is actually doing when your code runs. That knowledge makes you better in every language you learn after.

Who This Guide Is For

- College freshmen taking CS 101 or an equivalent introductory course
- Students interested in competitive programming (USACO, ICPC, LeetCode)
- Anyone who wants to understand memory management and systems concepts
- Students in data structures and algorithms courses who want to understand what is happening under the hood

A Realistic Expectation

C++ will frustrate you. You will see segfaults and wonder why your program compiled but crashed. That is normal. Work through it methodically. The students who stick with it gain instincts that are nearly impossible to build any other way.

The 8-Week Curriculum

Week 1 — Setup, Hello World, Variables, and Basic I/O

Goal: Get your environment running and understand the basics of a C++ program.

Topics:

- Installing a C++ compiler: GCC (via MinGW on Windows, Xcode on Mac, or your Linux package manager)
- Setting up VS Code with the C/C++ extension, or using an IDE like CLion

- Structure of a C++ program: `#include`, `using namespace std`, `int main()`
- `cout` and `cin` — printing output and reading input
- Primitive types: `int`, `double`, `float`, `char`, `bool`
- `string` (from `<string>`)
- Declaring and initializing variables
- Arithmetic operators
- `const` keyword

Practice Exercises:

1. Write a Hello World program. Compile and run it from the command line.
2. Declare variables for your name (`string`), age (`int`), and GPA (`double`). Print them all in one formatted sentence.
3. Read two integers from the user with `cin` and print their sum, difference, product, and quotient.
4. Write a program that calculates simple interest: `SI = (P * R * T) / 100`. Read P, R, and T from the user.
5. What is the difference between `int` and `double`? Try dividing 7 by 2 with each and explain the output.

Week 2 — Control Flow

Goal: Write programs that make decisions and repeat actions.

Topics:

- `if`, `else if`, `else`
- Comparison operators and logical operators (`&&`, `||`, `!`)
- `while` loops
- `for` loops
- `do-while` loops
- `break` and `continue`
- `switch` statements
- Nested loops

Practice Exercises:

1. Read an integer and print whether it is positive, negative, or zero

2. Print a right-angled triangle of stars: row 1 has 1 star, row 2 has 2, up to row 5 — use nested loops
3. Write a number guessing game: hardcode a target number, let the user guess, and give "too high" / "too low" / "correct" feedback in a loop
4. Write a program that prints the first N Fibonacci numbers (read N from the user)
5. Use a `while` loop to keep reading integers from the user until they enter 0, then print the total

---

Week 3 — Functions and Scope

Goal: Write organized, reusable code.

Topics:

- Defining and calling functions
- Return types and `void`
- Parameters and arguments
- Function prototypes (forward declarations)
- Pass by value
- Variable scope: local, global
- Recursion (intro — factorial and Fibonacci)

Practice Exercises:

1. Write a function `int square(int n)` and test it in `main`
2. Write a function `bool isPrime(int n)` — test it for several values
3. Write a recursive `int factorial(int n)` function. What happens when you call `factorial(20)`? What about `factorial(0)`?
4. Write a function `void printStars(int n)` that prints n stars on a single line. Use it to build the triangle from Week 2.
5. Write a function `double average(int arr[], int size)` that takes an array and its size and returns the average — we will revisit this in Week 4.

Week 4 — Arrays and Strings

Goal: Store sequences of data and work with text.

Topics:

- Declaring and initializing arrays

- Accessing elements by index
- Passing arrays to functions (and why the size must be passed separately)
- Common array operations: min, max, sum, reverse
- Multi-dimensional arrays (2D intro)
- C-style strings vs. `std::string`
- Common `string` methods: `length()`, `substr()`, `find()`, `at()`, concatenation

Practice Exercises:

1. Declare an array of 5 integers. Read them from the user, then print their sum and average.
2. Write a function `int findMax(int arr[], int size)` that returns the maximum value.
3. Write a function `void reverseArray(int arr[], int size)` that reverses the array in place.
4. Read a word from the user and print it reversed using a loop.
5. Count the number of vowels in a string entered by the user.
6. Create a 3x3 2D array representing a tic-tac-toe board. Initialize it with dots and print it as a grid.

Week 5 — Pointers and References

Goal: Understand how memory addresses work — one of C++'s most distinctive features.

Topics:

- What a pointer is: storing a memory address
- The address-of operator `&`
- The dereference operator `*`
- Pointer types: `int*`, `double*`, etc.
- `nullptr`
- Pass by reference vs. pass by value
- Reference variables (`int& ref = x`)
- Why pointers matter: efficiency, dynamic memory, data structures
- Array names as pointers (connection to Week 4)

Note: This week is foundational. It is also the week many beginners hit a wall. Go slowly. Draw diagrams showing memory addresses and what points where. This investment pays off heavily.

Practice Exercises:

1. Declare an integer, print its value and its memory address using `&`. Then declare a pointer to it and print the pointer's value and the dereferenced value.
2. Write a function `void swap(int* a, int* b)` that swaps two integers using pointers. Test it.
3. Rewrite `swap` using references instead of pointers: `void swap(int& a, int& b)`.
4. Write a function `void doubleAll(int arr[], int size)` that doubles every element in place (no return value needed — why?).
5. Draw on paper: what does memory look like when you have `int x = 5; int* p = &x;`? Label the addresses and values.

Week 6 — OOP: Classes and Objects

Goal: Model real-world entities as objects using classes.

Topics:

- Defining a class: `class`, member variables, member functions
- `public` vs. `private` access modifiers
- Constructors: default and parameterized
- Destructors (intro — what they are and when they run)
- Getters and setters
- `this` pointer
- Creating objects on the stack
- Separating class declaration (`.h`) from definition (`.cpp`) — intro

Practice Exercises:

1. Create a `Rectangle` class with private `width` and `height`, a constructor, and methods `getArea()` and `getPerimeter()`. Instantiate two rectangles and print their areas.
2. Create a `BankAccount` class with a private balance, and `deposit()`, `withdraw()`, and `getBalance()` methods. Add validation so withdrawal cannot exceed the balance.
3. Create a `Student` class with name and grade, and a method `isPassing()` that returns true if grade >= 60.
4. Create an array of three `Student` objects and print only the passing ones.

Week 7 — Vectors, Maps, and Sets (STL Basics)

Goal: Use the C++ Standard Template Library for common data structure needs.

Topics:

- Why the STL exists: you should not reinvent the wheel
- `vector<T>`: dynamic arrays, `push_back()`, `size()`, indexing, iteration
- `map<K, V>`: sorted key-value pairs, `insert()`, `find()`, `count()`, iteration
- `set<T>`: unique sorted elements, `insert()`, `find()`, `count()`
- Range-based for loops
- `#include <vector>`, `#include <map>`, `#include <set>`
- Brief intro to `algorithm` header: `sort()`, `find()`, `max_element()`

Practice Exercises:

1. Create a `vector<int>`, read 5 integers into it, sort it using `std::sort()`, and print the sorted result.
2. Write a word frequency counter using a `map<string, int>` — read words until the user enters "done" and then print each word and its count.
3. Read 10 integers from the user and use a `set<int>` to find and print the unique values.
4. Refactor your `Student` array from Week 6 to use a `vector<Student>`.

Week 8 — Memory Management Basics

Goal: Understand the difference between stack and heap, and why it matters.

Topics:

- Stack vs. heap: what they are and how they differ
- Dynamic memory allocation: `new` and `delete`
- `new` for arrays: `new int[10]` and `delete[]`
- Memory leaks: what they are and why they are bad
- Dangling pointers: what happens when you use memory after freeing it
- Smart pointers: `unique_ptr` and `shared_ptr` (intro — know they exist)
- Why modern C++ prefers RAII and smart pointers over raw `new`/`delete`
- Connecting back to Week 5: pointers are required to use heap memory

Note: Most beginners do not need to use `new`/`delete` directly in their first projects — prefer `vector` and `string` which manage memory for you. This week is about understanding what those containers are doing internally.

Practice Exercises:

1. Allocate an array of 5 integers on the heap using `new`. Fill it, print it, then free it with `delete[]`.
2. What happens if you `delete` the same pointer twice? Try it (carefully) and read the error. You do not need to fix it — just observe.
3. Write a function that allocates a `new int`, sets its value, and returns the pointer. In `main`, use the value and then `delete` it.
4. Rewrite Exercise 3 using `unique_ptr<int>` from `<memory>`. Notice you do not need to call `delete`.
5. In your own words: why would you ever use heap memory instead of just declaring a variable normally?

Resources

Official and Core

- cppreference.com — the definitive C++ reference, bookmark it
- LearnCpp.com — the best free online C++ tutorial, extremely thorough

Practice Platforms

- HackerRank — C++ — good for syntax and beginner challenges
- Codeforces — competitive programming, great for algorithmic thinking
- USACO Guide — if you are interested in competitive programming at the high school level
- LeetCode — filter for Easy, solve in C++

Books

- *Programming: Principles and Practice Using C++* by Bjarne Stroustrup — written by the creator of C++, accessible and thorough
- *C++ Primer* by Lippman, Lajoie, Moo — comprehensive introduction, commonly used in university courses

---

Why This Makes You a Better Programmer

Here is the honest reason to learn C++: it removes the mystery. In Python, a list just works. In Java, an ArrayList just works. In C++, you will learn what is happening inside those abstractions — how memory is allocated, why certain operations are fast and others are slow, what a pointer actually is.

When you later return to higher-level languages with that knowledge, you write better code. You understand why copying a large object is expensive, why passing by reference matters, and why you

should not store large objects in tight loops. These are the kinds of insights that separate good programmers from great ones.

Written by Parth Shah. For tutoring inquiries, visit ajconsultation.com.