# 📘 *C++ for Clever Kids!*

## ✨ Table of Contents



## Chapter 1: Hello, Computer!

### 👩‍💻 Learn It:

When we talk to a computer, we use a special language. One of these is called **C++**. Just like how we say "Hello!" to people, we can teach a computer to say "Hello!" too.

### 👀 See It:

Here's what that looks like in C++:

```
#include <iostream>

int main() {
    std::cout << "Hello, world!" << std::endl;
    return 0;
```

```
}
```

🧠 Do It:

- Can you say what this program does?
- Try changing the word *"world"* to *your name*. What happens?

✍️ Write your own message for the computer to say! Example:
`"Hello, I'm a coding hero!"`

---

## Chapter 2: What Are Numbers (and Friends)?

👩‍💻 **Learn It:**

Computers love to remember things — like numbers, words, or whether something is true or false. In C++, we give these things names so we can use them later. These are called **variables**. Each one has a **type** — like a kind of box that holds a certain thing.

Let's meet the most fun types!

👀 See It:

```
int number = 1;            // Whole number (like 1, 2, 3)

float f = 0.1f;            // A small number with a dot
double pi = 3.1415926;     // A bigger number with a dot (more
precise!)
std::string s = "Hi";      // A word or sentence
char c = 'H';              // Just one letter
bool b = true;             // true or false (yes or no!)

const int lucky = 7;       // A number that never changes!
```

🧠 Do It:

- Try changing `number` to your favourite number!
- Can you make a `string` with your name?
- What happens if you set `bool b = false;`?
- Make a `const` called `birthYear` and give it your year of birth.

```
const int birthYear = 2017;
```

---

## Chapter 3: Talking with the Computer 👂💬

👩‍🏫 **Learn It:**

Sometimes, we want the computer to listen to *us*! With C++, we can ask the computer to wait for us to **type something**, and then it can **remember** what we typed.

👀 See It:

```
#include <iostream>

int main() {
    int num;

    std::cout << "Type a number: ";
    std::cin >> num;

    std::cout << "You entered " << num;
    return 0;
}
```

What's happening here?

- `std::cout` → Says something *to you*
- `std::cin` → Waits for *you* to type something

🧠 Do It:

- Change `int` to `std::string` and type your name!
- Ask the computer to say: *"Nice to meet you, <your name>!"*
- Try asking the user for their age and say something back!

Bonus challenge:

```
std::cout << "How old are you? ";
std::cin >> age;
std::cout << "Wow! " << age << " years old!";
```

---

## Chapter 4: Lists of Things (Arrays!) 🧺

👩‍🏫 **Learn It:**

What if we have *lots of numbers* — like test scores? Instead of making many variables, we can use a **list** called an **array**! It's like a row of boxes, and each box has a number (called an **index**) that helps us find it.

👀 See It:

```
#include <iostream>
#include <array>

int main() {
    std::array<int, 3> marks; // Just made the boxes
    marks[0] = 92;
    marks[1] = 97;
    marks[2] = 98;
```

```
    std::cout << marks[0]; // Says: 92
}
```

You can also **fill it in one go!**

```
std::array<int, 3> marks = {92, 97, 98};
```

Forgot a number? It becomes **0**!

```
std::array<int, 3> marks = {92, 97};
std::cout << marks[2]; // Says: 0
```

Let's see it like a row:

```
┌──────┬──────┬──────┐
|  92  |  97  |  98  |
└──────┴──────┴──────┘
   [0]   [1]   [2]
```

Now with more scores:

```
std::array<int, 6> marks = {92, 97, 98, 99, 98, 94};

std::cout << marks[0]; // First box
marks[1] = 99;          // Fix second score!
std::cin >> marks[2];   // You type the third!
```

🌟 Arrays can hold **letters** too:

```
char ref[5] = {'R', 'e', 'f'};
```

We can read each letter like this:

```
for (const char &n : ref) {
    std::cout << std::string(1, n);
}
```

Or the classic way:

```
for (int i = 0; i < 5; ++i) {
    std::cout << ref[i];
}
```

Even better... **Arrays in rows AND columns!** 🧮

```
int x[2][6] = {
    {1,2,3,4,5,6},
    {6,5,4,3,2,1}
};

for (int i = 0; i < 2; ++i) {
    for (int j = 0; j < 6; ++j) {
        std::cout << x[i][j] << " ";
    }
}
```

This prints:

```
1 2 3 4 5 6 6 5 4 3 2 1
```

🧠 Do It:

- Make a list of your 3 favorite numbers.
- Change one to something new.
- Print them all using a loop!
- Try making a table with 2 rows and 3 columns.

---

## Chapter 5: If This, Then That (Decisions!) 🤔

## 👩‍💻 Learn It:

Sometimes, the computer needs to *choose*. We can teach it to decide using **if**, **else**, and other fun tools!

### 👀 See It:

#### 🧪 The `if` Statement

```
int number = 16;

if (number % 2 == 0) {
    std::cout << "Even!";
} else {
    std::cout << "Odd!";
}
```

💡 % means "remainder" — 16 ÷ 2 has no leftovers, so it's even!

---

#### 🧪 Else If: So Many Choices!

```
int score = 99;

if (score == 100) {
    std::cout << "Superb!";
} else if (score >= 90) {
    std::cout << "Excellent!";
} else if (score >= 80) {
    std::cout << "Very Good!";
} else if (score >= 70) {
    std::cout << "Good!";
} else if (score >= 60) {
    std::cout << "OK!";
} else {
    std::cout << "Hmm... Try again!";
}
```

---

### 🎲 Useful Operators

**Relational:**

| Code | Means |
|---|---|
| a == b | a equals b |
| a != b | a does *not* equal b |
| a > b | a is bigger than b |
| a < b | a is smaller than b |
| a >= b | a is big or equal to b |
| a <= b | a is small or equal to b |

**Assignment Shortcuts:**

```
x += 2; // same as x = x + 2
if (sunny && warm) { ... }  // AND
if (raining || cold) { ... } // OR
if (!hungry) { ... } // NOT
```

---

### 🧪 Ternary: Tiny if-else!

```
int x = 3, y = 5, max;
max = (x > y) ? x : y;
std::cout << max; // Says: 5
```

Same as:

```
if (x > y) {
    max = x;
} else {
    max = y;
}
```

---

### 🎮 Switch: Choose from a Menu!

```
int num = 2;

switch (num) {
    case 0: std::cout << "Zero"; break;
    case 1: std::cout << "One"; break;
    case 2: std::cout << "Two"; break;
    case 3: std::cout << "Three"; break;
    default: std::cout << "What?"; break;
}
```

---

🧠 Do It:

- Ask the user for a number and say if it's odd or even!
- Use `else if` to grade scores like a teacher.
- Try out a `switch` to print a day of the week (0 = Sunday, 1 = Monday...).

---

## Chapter 6: Let's Repeat (Loops!) 🔁

👩‍🏫 **Learn It:**

What if we want the computer to do something *again* and *again*? That's where **loops** come in! A loop repeats code until we say "stop!"

---

🌀 While Loop

Repeats while something is *true*.

```
int i = 0;
while (i < 6) {
    std::cout << i++;
}
// Output: 012345
```

---

### 🌀 Do-While Loop

This loop always runs *at least once*!

```
int i = 1;
do {
    std::cout << i++;
} while (i <= 5);
// Output: 12345
```

---

### 🔄 For Loop

Great for counting!

```
for (int i = 0; i < 5; i++) {
    std::cout << i;
}
// Output: 01234
```

You can even do **two things at once**:

```
for (int i = 0, j = 2; i < 3; i++, j--) {
    std::cout << "i=" << i << ", j=" << j << "; ";
}
// Output: i=0, j=2; i=1, j=1; i=2, j=0;
```

---

### ⏩ Skip with `continue`

```
for (int i = 0; i < 10; i++) {
    if (i % 2 == 0) continue;
    std::cout << i;
}
// Output: 13579
```

---

### ⛔ Stop with `break`

```
int password, tries = 0;
while (password != 1234) {
    if (tries++ >= 3) {
        std::cout << "Locked!\n";
        break;
    }
    std::cout << "Password: ";
    std::cin >> password;
}
```

---

### 🔁 Forever Loops (Be Careful!)

```
while (true) {
    std::cout << "infinite loop";
}

for (;;) {
    std::cout << "infinite loop";
}
```

---

### 🌟 Range-Based Loops (C++11)

```
for (int n : {1, 2, 3, 4, 5}) {
    std::cout << n << " ";
}
// Output: 1 2 3 4 5
```

Even works on **words**:

```
std::string hello = "Hello";
for (char c : hello) {
    std::cout << c << " ";
}
// Output: H e l l o
```

---

✨ `for_each` Magic (C++11)

```
#include <array>
#include <algorithm>

auto print = [](int num) { std::cout << num << std::endl; };
std::array<int, 4> arr = {1, 2, 3, 4};
std::for_each(arr.begin(), arr.end(), print);
```

---

🧠 Do It:

- Count from 1 to 10 with a `while` loop!
- Make a `for` loop that prints only odd numbers.
- Use a `break` to stop the loop after 3 tries.
- Try a `range-based` loop to spell your name!

---

## Chapter 7: Make Your Own Magic (Functions!) 🪄

👩‍🏫 Learn It:

A **function** is like a mini-program that does one job. You give it something (called **arguments**) and it can give something back (**return value**).

---

✨ Make a Function

```
int add(int a, int b) {
    return a + b;
}
```

This `add` function takes **two numbers** and gives you the **sum**!

You use it like this:

```
std::cout << add(10, 20); // Output: 30
```

---

✨ Overloading: Same Name, Different Jobs

C++ lets you make **many functions with the same name**, as long as they look different!

```
void fun(std::string a, std::string b) {
    std::cout << a + " " + b;
}

void fun(std::string a) {
    std::cout << a;
}

void fun(int a) {
    std::cout << a;
}
```

You can now call:

```
fun("Hello");           // One word
fun("Hello", "World");  // Two words
fun(123);               // A number!
```

---

⚙️ Built-in Functions (They Come With C++!)

```
#include <cmath>

std::cout << sqrt(9);  // Square root: 3
std::cout << pow(2, 3); // 2 to the power of 3 = 8
```

---

🧠 Do It:

- Write a function that doubles a number.

- Write one that returns your name.
- Try using `sqrt()` or `pow()` from `<cmath>`.
- Make your own overloaded `greet()` function!

```
greet("Alex");
greet("Alex", "the Brave");
```

---

## Chapter 8: Let's Build Something Big! (Classes & Objects) 🏗️

👩‍🏫 Learn It:

Classes are like **blueprints** — they describe what something *is* and what it *can do*. When you use a class to make a real thing, that's called an **object**.

---

🧱 A Simple Class

```
class MyClass {
  public:
    int myNum;
    std::string myString;
};
```

This class has **two things**:

- A number (`myNum`)
- A word or message (`myString`)

---

🎁 Make an Object (Your Own Copy!)

```
MyClass myObj;
myObj.myNum = 15;
myObj.myString = "Hello";
```

```
std::cout << myObj.myNum << std::endl;     // 15
std::cout << myObj.myString << std::endl;  // Hello
```

---

🏗️ Constructors: Auto-Build

A **constructor** is a special function that runs when an object is made!

```
class MyClass {
  public:
    int myNum;
    std::string myString;

    MyClass() {
      myNum = 0;
      myString = "";
    }
};
```

---

💥 Destructors: Auto-Clean

A **destructor** runs when your object goes away.

```
~MyClass() {
  std::cout << "Object destroyed." << std::endl;
}
```

---

🛠️ Class Methods (Class Actions!)

```
class MyClass {
  public:
    void sayHi() {
      std::cout << "Hello World!" << std::endl;
    }
};
```

```
MyClass obj;
obj.sayHi();  // Hello World!
```

---

🔒 Access Modifiers

```
class MyClass {
  public:    // Everyone can see this
    int x;

  private:   // Only the class can see this
    int y;

  protected: // The class and its "children" can see
    int z;
};

myObj.x = 25;  // OK!
myObj.y = 50;  // ❌ Error! It's private
```

---

🔍 Getters & Setters (Safe Access!)

```
class MyClass {
  private:
    int myNum;

  public:
    void setMyNum(int num) {
      myNum = num;
    }
    int getMyNum() {
      return myNum;
    }
};
```

---

👪 Inheritance (Family of Classes!)

```
class Vehicle {
```

```
  public:
    std::string brand = "Ford";
    void honk() {
      std::cout << "Tuut, tuut!" << std::endl;
    }
};

class Car : public Vehicle {
  public:
    std::string model = "Mustang";
};

Car myCar;
myCar.honk();  // Tuut, tuut!
std::cout << myCar.brand + " " + myCar.model;
```

---

🧠 Do It:

- Make a class `Animal` with a `name` and a `sound()`.
- Create a `Dog` class that *inherits* from `Animal`.
- Make your dog say “Woof!”

---

## Chapter 9: The Secret Wizard! (Preprocessor) 🧙‍♂️

👩‍🏫 **Learn It:**

Before your C++ code runs, a **special helper** called the **preprocessor** looks at it first! It handles things like including files, making shortcuts, or even hiding code.

It starts with `#` and runs *before* the real code starts.

---

📦 Includes

```
#include <iostream>  // From C++ system
```

```
#include "myfile.h"  // From your folder
```

This tells C++: “Bring in this file before starting!”

---

### 🪄 Defines (Magic Shortcuts!)

```
#define PI 3.14

std::cout << PI;  // Becomes: std::cout << 3.14;
```

You can also *undefine* something:

```
#undef PI
```

---

### 🧠 Conditions with `#if`, `#ifdef`, `#ifndef`

```
#define DEBUG

#ifdef DEBUG
  std::cout << "Debug mode on!";
#elif defined VERBOSE
  std::cout << "Verbose mode!";
#else
  std::cout << "Normal mode.";
#endif
```

---

### ⛔ Errors and Warnings

```
#define VERSION 2

#if VERSION == 2
  #error "Version 2 not supported!"
  // #warning "Are you sure?"  ← Not standard C++, may depend on
compiler
#endif
```

---

⚙️ Macros: Mini-Machines!

```
#define DEG(x) ((x) * 57.29)

std::cout << DEG(3.14);  // Becomes: ((3.14) * 57.29)
```

---

🔗 Glue Words (Token Concatenation)

```
#define MAKE_NAME(x) x##_cat

MAKE_NAME(my);  // Becomes: my_cat
```

---

🔤 Turn Words into Strings

```
#define STR(x) #x

std::string name = STR(Hello);  // "Hello"
```

---

📁 Where Did That Happen?

```
#define LOG(msg) std::cout << __FILE__ << ":" << __LINE__ << " " << msg;

LOG("Oops!");
// Output: main.cpp:42 Oops!
```

---

🔎 Even Smarter!

```
#if __has_include(<vector>)
  #include <vector>
#endif
```

---

🧠 Do It:

- Try `#define AGE 8` and print it.
- Create a macro that doubles a number.
- Use `#ifdef DEBUG` to print a message.

---

## Chapter 10: Quick Extras (Little Big Things!) 🌈

👩‍🏫 **Learn It:**

C++ has a *lot* of hidden treasures — some are tiny tricks, some are powerful tools. Let's meet a few!

---

🔡 Escape Sequences

Used in strings to **do something special**:

| Code | What it does |
|---|---|
| `\n` | New line |
| `\t` | Tab (like big space) |
| `\\` | Backslash |
| `\"` | Double quote |
| `\'` | Single quote |
| `\b` | Backspace (⬅️) |
| `\0` | End of a string |

```
std::cout << "Hello\nWorld";  // Hello (new line) World
```

---

## 🗝️ Keywords — Magic Words You Can't Use as Names!

C++ has **reserved words** you must not use for variables or functions.

Examples:

- `if`, `else`, `while`, `return`
- `int`, `float`, `bool`, `class`
- `private`, `public`, `virtual`

Just a few fancy ones:

- `constexpr` – Known during compile-time
- `noexcept` – A function that doesn't throw
- `mutable` – Can change even in const object
- `co_await` / `co_yield` – For coroutines (advanced!)

📜 Full keyword list? You've got it. You already shared it perfectly above!

---

## 🧵 Preprocessor Recap

Remember those `#include`, `#define`, `#ifdef`, and even cool things like:

```
#if __has_include(<vector>)
    #include <vector>
#endif
```

The preprocessor helps get your code **ready to compile**!

---

🧠 Do It:

- Use escape characters to format your message.
- Try printing out a full ASCII art box using `\n` and `\t`.
- Look through the keyword list — how many have you already used?