

C++ for Absolute Beginners

Solutions to All 30 Exercises — Organised by Topic

Each solution includes a complete working program and an explanation of the key concepts. Try to solve each exercise yourself first!

Table of Contents

Section 1 — Very Basics (solutions 1–9)

- 01. Hello, World! (Output) ★■■■
- 02. Print Your Name (Output) ★■■■
- 03. Sum of Two Numbers (Variables) ★■■■
- 04. Read Two Integers (Input) ★■■■
- 05. int vs float Division (Variable Types) ★■■■
- 06. double Precision Pi (Variable Types) ★■■■
- 07. Overflow Demonstration (Variable Types) ★★ ■
- 08. char Arithmetic (Variable Types) ★★ ■
- 09. Type Conversion Chain (Variable Types) ★★★

Section 2 — Operators & Syntax (solutions 10–30)

- 10. Even or Odd (Conditionals) ★■■■
- 11. Maximum of Two Numbers (Conditionals) ★■■■
- 12. Leap Year Checker (Logical Operators) ★★ ■
- 13. Login Gate (Logical Operators) ★★ ■
- 14. PIN Change Validator (Logical Operators) ★★ ■
- 15. Countdown with a Loop (Loops (for)) ★★ ■
- 16. Sum 1 to N (Loops) ★★ ■
- 17. Multiplication Table (Loops) ★★ ■
- 18. Factorial (Loops) ★★ ■
- 19. Guess the Number (Loops (while)) ★★ ■
- 20. Collatz Conjecture (Loops (while)) ★★★
- 21. FizzBuzz (Conditionals) ★★ ■
- 22. Reverse a String (Strings) ★★ ■
- 23. Count Vowels (Strings) ★★ ■
- 24. Palindrome Check (Strings) ★★★
- 25. Simple Array Average (Arrays) ★★ ■
- 26. Find Minimum in Array (Arrays) ★★ ■
- 27. Bubble Sort (Arrays) ★★★
- 28. Basic Struct — Student Record (Structs) ★★ ■
- 29. Array of Structs — Temperature Log (Structs) ★★★
- 30. Struct with Comparison (Structs) ★★★

Section 1 — Very Basics

Solution 1 of 30

Hello, World!



Topic: Output / Program Structure

Solution Code

```
#include <iostream>

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

Explanation

Every C++ program needs a `main()` function. `#include` brings in the standard input/output library, which gives us `std::cout` for printing to the screen and `std::endl` to end the line and flush the buffer.

Print Your Name



Topic: Output / Variables

Solution Code

```
#include <iostream>
#include <string>

int main() {
    std::string name = "Alice";
    std::cout << "My name is " << name << "." << std::endl;
    return 0;
}
```

Explanation

std::string stores text. We concatenate it with << in the output stream.

Sum of Two Numbers



Topic: Variables / Arithmetic

Solution Code

```
#include <iostream>

int main() {
    int a = 7;
    int b = 3;
    int sum = a + b;
    std::cout << sum << std::endl;
    return 0;
}
```

Explanation

int holds whole numbers. The + operator adds them. We store the result in a third variable.

Read Two Integers



Topic: Input / Output

Solution Code

```
#include <iostream>

int main() {
    int a, b;
    std::cout << "Enter two numbers: ";
    std::cin >> a >> b;
    std::cout << "Sum: " << a + b << std::endl;
    return 0;
}
```

Explanation

std::cin reads from the keyboard. The >> operator chains nicely for multiple inputs.

int vs float Division



Topic: Variable Types / int / float

Solution Code

```
#include <iostream>
#include <iomanip>

int main() {
    int a = 7;
    int b = 2;

    // Integer division: decimal part is discarded
    std::cout << "Integer division: " << a / b << std::endl;

    // Float division: cast one operand first
    std::cout << "Float division: " << (float)a / b << std::endl;

    return 0;
}
```

Explanation

When both operands are int, C++ performs integer division and drops the remainder. Casting just one side to float is enough to switch the entire expression to floating-point division.

double Precision Pi



Topic: Variable Types / double

Solution Code

```
#include <iostream>
#include <iomanip>

int main() {
    float fpi = 3.14159265358979f;
    double dpi = 3.14159265358979;

    std::cout << std::fixed << std::setprecision(10);
    std::cout << "float pi: " << fpi << std::endl;
    std::cout << "double pi: " << dpi << std::endl;

    return 0;
}
```

Explanation

float has ~7 significant digits of precision; double has ~15-16. The 'f' suffix on the float literal tells the compiler to treat it as a float constant rather than a double. `std::fixed` and `std::setprecision(10)` force 10 decimal places in output.

Overflow Demonstration



Topic: Variable Types / int / long long

Solution Code

```
#include <iostream>

int main() {
    int i = 2147483647;
    i = i + 1; // wraps around!
    std::cout << "int overflow: " << i << std::endl;

    long long ll = 2147483647LL;
    ll = ll + 1; // safe
    std::cout << "long long result: " << ll << std::endl;

    return 0;
}
```

Explanation

int is typically 32 bits. Adding 1 to its maximum value causes undefined behaviour that in practice wraps to the most negative value (-2147483648). *long long* is 64 bits and handles this range comfortably. The LL suffix marks the literal as long long.

char Arithmetic



Topic: Variable Types / char / int

Solution Code

```
#include <iostream>

int main() {
    char c = 'A';

    std::cout << "Character: " << c << std::endl;
    std::cout << "ASCII value: " << (int)c << std::endl;
    std::cout << "3 letters later: " << (char)(c + 3) << std::endl;

    return 0;
}
```

Explanation

char stores a single character as its ASCII integer code. 'A' is 65. Casting to int reveals the number; adding 3 gives 68 which is 'D'. Casting back to char converts the integer to the corresponding character.

Type Conversion Chain



Topic: Variable Types / int / float / double / char

Solution Code

```
#include <iostream>

int main() {
    double d = 9.99;
    float f = d; // double -> float: small rounding
    int i = f; // float -> int: truncates decimal
    char ch = i; // int -> char: stores ASCII code 9

    std::cout << "double: " << d << std::endl;
    std::cout << "float: " << f << std::endl;
    std::cout << "int: " << i << std::endl;
    std::cout << "char (as int): " << (int)ch << std::endl;

    return 0;
}
```

Explanation

Each narrowing conversion loses information. double -> float: minor rounding. float -> int: truncation (9.99 becomes 9, not 10). int -> char: stores the raw integer value 9 as a byte (ASCII 9 is a non-printable tab-like character, so we cast back to int to print the number). This chain illustrates why implicit narrowing conversions can introduce subtle bugs.

Section 2 — Operators & Syntax

Solution 10 of 30

Even or Odd



Topic: Conditionals

Solution Code

```
#include <iostream>

int main() {
    int n;
    std::cout << "Enter a number: ";
    std::cin >> n;
    if (n % 2 == 0) {
        std::cout << "Even" << std::endl;
    } else {
        std::cout << "Odd" << std::endl;
    }
    return 0;
}
```

Explanation

The % (modulo) operator gives the remainder. If remainder is 0, the number is even.

Maximum of Two Numbers



Topic: Conditionals

Solution Code

```
#include <iostream>

int main() {
    int a, b;
    std::cout << "Enter two numbers: ";
    std::cin >> a >> b;
    if (a >= b) {
        std::cout << "Max: " << a << std::endl;
    } else {
        std::cout << "Max: " << b << std::endl;
    }
    return 0;
}
```

Explanation

Using >= handles the equal case naturally — either value would be correct.

Leap Year Checker



Topic: Logical Operators / && ||

Solution Code

```
#include <iostream>

int main() {
    int year;
    std::cout << "Year: ";
    std::cin >> year;

    bool isLeap = (year % 4 == 0) && (year % 100 != 0 || year % 400 == 0);

    if (isLeap) {
        std::cout << "Leap year" << std::endl;
    } else {
        std::cout << "Not a leap year" << std::endl;
    }

    return 0;
}
```

Explanation

The && requires BOTH sides to be true. The || inside the parentheses means 'not a century year, OR divisible by 400'. 2000 passes (divisible by 400); 1900 fails (century year, not divisible by 400). Storing the result in a bool variable keeps the if statement readable.

Login Gate



Topic: Logical Operators / && ||

Solution Code

```
#include <iostream>

int main() {
    const std::string correctUser = "admin";
    const std::string correctPass = "cpp123";

    std::string user, pass;
    std::cout << "Username: ";
    std::cin >> user;
    std::cout << "Password: ";
    std::cin >> pass;

    bool userOk = (user == correctUser);
    bool passOk = (pass == correctPass);

    if (userOk && passOk) {
        std::cout << "Access granted" << std::endl;
    } else {
        if (!userOk) {
            std::cout << "Warning: unknown user" << std::endl;
        } else if (userOk && !passOk) {
            std::cout << "Warning: wrong password" << std::endl;
        }
        std::cout << "Access denied" << std::endl;
    }

    return 0;
}
```

Explanation

Breaking the conditions into bool variables (*userOk*, *passOk*) makes the logic easier to read and reuse. *&&* ensures both must be true for access. The *!* (NOT) operator inverts a bool — *!userOk* means 'username is wrong'. This structure also avoids revealing which field was wrong to an attacker unless you deliberately choose to (as we do here for teaching purposes).

PIN Change Validator



Topic: Logical Operators / !=

Solution Code

```
#include <iostream>

int main() {
    std::string currentPin, newPin, confirm;

    std::cout << "Current PIN: ";
    std::cin >> currentPin;
    std::cout << "New PIN: ";
    std::cin >> newPin;

    if (newPin == currentPin) {
        std::cout << "Error: new PIN must differ from current PIN"
        << std::endl;
        return 1;
    }

    std::cout << "Confirm PIN: ";
    std::cin >> confirm;

    if (confirm != newPin) {
        std::cout << "Error: PINs do not match" << std::endl;
        return 1;
    }

    std::cout << "PIN changed successfully" << std::endl;
    return 0;
}
```

Explanation

!= is the 'not equal to' operator — the direct opposite of ==. Here it catches a mistyped confirmation entry. Returning 1 from main signals an error to the operating system (0 means success). Checking newPin == currentPin first (before asking for confirmation) saves the user an unnecessary step when they reuse their old PIN.

Countdown with a Loop



Topic: Loops (for)

Solution Code

```
#include <iostream>

int main() {
    for (int i = 10; i >= 1; i--) {
        std::cout << i << std::endl;
    }
    std::cout << "Blast off!" << std::endl;
    return 0;
}
```

Explanation

The for loop has three parts: init (i=10), condition (i>=1), and update (i--). i-- decrements by 1 each iteration.

Sum 1 to N



Topic: Loops / Accumulator

Solution Code

```
#include <iostream>

int main() {
    int n;
    std::cout << "N = ";
    std::cin >> n;
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        sum += i;
    }
    std::cout << "Sum = " << sum << std::endl;
    return 0;
}
```

Explanation

An accumulator (sum) starts at 0. Each loop iteration adds the current i to sum using +=.

Multiplication Table



Topic: Loops / Nested

Solution Code

```
#include <iostream>

int main() {
    for (int i = 1; i <= 5; i++) {
        for (int j = 1; j <= 5; j++) {
            std::cout << i << "x" << j << "=" << i * j;
            if (j < 5) std::cout << " ";
        }
        std::cout << std::endl;
    }
    return 0;
}
```

Explanation

Nested loops: the outer loop picks each row (i), the inner loop fills each column (j). The inner loop completes fully before i increments.

Factorial



Topic: Loops / Math

Solution Code

```
#include <iostream>

int main() {
    int n;
    std::cout << "n = ";
    std::cin >> n;
    long long result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    std::cout << n << "! = " << result << std::endl;
    return 0;
}
```

Explanation

*We use long long to hold larger results. result *= i is shorthand for result = result * i.*

Guess the Number



Topic: Loops (while) / Input

Solution Code

```
#include <iostream>

int main() {
    int secret = 42;
    int guess;
    int count = 0;

    while (true) {
        std::cout << "Guess: ";
        std::cin >> guess;
        count++;

        if (guess == secret) {
            std::cout << "Correct! You got it in "
            << count << " guesses." << std::endl;
            break;
        } else if (guess < secret) {
            std::cout << "Too low!" << std::endl;
        } else {
            std::cout << "Too high!" << std::endl;
        }
    }

    return 0;
}
```

Explanation

while (true) creates an infinite loop that only exits via *break*. A count variable increments each iteration to track guesses. This pattern — loop forever, break on success — is very common for input-validation loops.

Collatz Conjecture



Topic: Loops (while) / Math

Solution Code

```
#include <iostream>

int main() {
    long long n;
    std::cout << "n = ";
    std::cin >> n;

    int steps = 0;
    std::cout << n;

    while (n != 1) {
        if (n % 2 == 0) {
            n = n / 2;
        } else {
            n = 3 * n + 1;
        }
        std::cout << " " << n;
        steps++;
    }

    std::cout << std::endl;
    std::cout << "Steps: " << steps << std::endl;
    return 0;
}
```

Explanation

We use `long long` because $3*n+1$ can temporarily grow very large even for modest starting values. The first value is printed before the loop; each subsequent value is printed inside it. The Collatz conjecture states this always reaches 1 — it has been verified for enormous numbers but never formally proved.

FizzBuzz



Topic: Conditionals / Loops

Solution Code

```
#include <iostream>

int main() {
    for (int i = 1; i <= 30; i++) {
        if (i % 15 == 0) {
            std::cout << "FizzBuzz";
        } else if (i % 3 == 0) {
            std::cout << "Fizz";
        } else if (i % 5 == 0) {
            std::cout << "Buzz";
        } else {
            std::cout << i;
        }
        std::cout << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

Explanation

Check for 15 first — if you check 3 and 5 first, a multiple of 15 would match one of those instead of printing FizzBuzz.

Reverse a String



Topic: Strings

Solution Code

```
#include <iostream>

int main() {
    std::string s;
    std::cout << "Enter a word: ";
    std::cin >> s;
    for (int i = s.length() - 1; i >= 0; i--) {
        std::cout << s[i];
    }
    std::cout << std::endl;
    return 0;
}
```

Explanation

s.length() gives the number of characters. We index from length-1 down to 0, printing each character.

Count Vowels



Topic: Strings / Characters

Solution Code

```
#include <iostream>

int main() {
    std::string text;
    std::cout << "Enter text: ";
    std::getline(std::cin, text);
    int count = 0;
    for (char c : text) {
        char lower = tolower(c);
        if (lower=='a' || lower=='e' || lower=='i'
            || lower=='o' || lower=='u') {
            count++;
        }
    }
    std::cout << "Vowels: " << count << std::endl;
    return 0;
}
```

Explanation

std::getline reads a full line including spaces. The range-based for loop (*for char c : text*) visits each character. *tolower()* normalises case.

Palindrome Check



Topic: Strings / Logic

Solution Code

```
#include <iostream>

int main() {
    std::string word;
    std::cout << "Enter a word: ";
    std::cin >> word;
    std::string rev = word;
    int n = word.length();
    for (int i = 0; i < n / 2; i++) {
        std::swap(rev[i], rev[n - 1 - i]);
    }
    if (word == rev) {
        std::cout << "Yes" << std::endl;
    } else {
        std::cout << "No" << std::endl;
    }
    return 0;
}
```

Explanation

We copy the word, then swap characters from the outside in to build its reverse. Comparing the original with the reversed copy tells us if it's a palindrome.

Simple Array Average



Topic: Arrays

Solution Code

```
#include <iostream>
#include <string>

int main() {
    int arr[] = {4, 8, 15, 16, 23};
    int size = 5;
    int sum = 0;
    for (int i = 0; i < size; i++) {
        sum += arr[i];
    }
    double avg = (double)sum / size;
    std::cout << "Average: " << avg << std::endl;
    return 0;
}
```

Explanation

Casting sum to double before dividing avoids integer division truncation. (double)sum / size gives 13.2, not 13.

Find Minimum in Array



Topic: Arrays

Solution Code

```
#include <iostream>
#include <string>

int main() {
    int arr[5];
    std::cout << "Enter 5 numbers: ";
    for (int i = 0; i < 5; i++) {
        std::cin >> arr[i];
    }
    int minVal = arr[0];
    for (int i = 1; i < 5; i++) {
        if (arr[i] < minVal) {
            minVal = arr[i];
        }
    }
    std::cout << "Min: " << minVal << std::endl;
    return 0;
}
```

Explanation

We seed minVal with the first element, then compare every remaining element. If we find something smaller, we update minVal.

Bubble Sort



Topic: Arrays / Sorting

Solution Code

```
#include <iostream>
#include <string>

int main() {
    int arr[] = {5, 3, 8, 1, 9, 2};
    int n = 6;
    std::cout << "Unsorted: ";
    for (int i = 0; i < n; i++)
        std::cout << arr[i] << " ";
    std::cout << std::endl;

    // Bubble sort
    for (int i = 0; i < n - 1; i++) { for (int j = 0; j < n - 1 - i; j++) { if (arr[j]
    > arr[j + 1]) {
        std::swap(arr[j], arr[j + 1]);

    }
    }
    }

    std::cout << "Sorted: ";
    for (int i = 0; i < n; i++)
        std::cout << arr[i] << " ";
    std::cout << std::endl;
    return 0;
}
```

Explanation

Each pass of the outer loop 'bubbles' the largest unsorted element to the end. The inner loop shrinks by i each time because the last i elements are already sorted.

Basic Struct — Student Record



Topic: Structs / Definition / Members

Solution Code

```
#include <iostream>
#include <string>

struct Student {
    std::string name;
    int age;
    double gpa;
};

int main() {
    Student students[3] = {
        {"Alice", 20, 3.8},
        {"Bob", 22, 3.5},
        {"Carol", 21, 3.9}
    };

    for (int i = 0; i < 3; i++) {
        std::cout << "Name: " << students[i].name
        << " Age: " << students[i].age
        << " GPA: " << students[i].gpa
        << std::endl;
    }

    Student best = students[0];
    for (int i = 1; i < 3; i++)
        if (students[i].gpa > best.gpa)
            best = students[i];

    std::cout << "Top student: " << best.name
    << " (" << best.gpa << ")" << std::endl;
    return 0;
}
```

Explanation

A struct bundles related data under one name. Each field is accessed with the dot operator. Structs can be assigned with = and stored in arrays just like plain types. The best-student search seeds with the first element then compares the rest — the same pattern as finding a minimum or maximum in a plain array.

Array of Structs — Temperature Log



Topic: Structs / Arrays / Statistics

Solution Code

```
#include <iostream>
#include <iomanip>

struct Reading {
    int hour;
    double temperature;
};

int main() {
    Reading log[5] = {
        {6, 12.5},
        {9, 17.3},
        {12, 22.1},
        {15, 20.8},
        {18, 15.4}
    };

    double sum = 0.0;
    int maxIdx = 0;

    for (int i = 0; i < 5; i++) {
        std::cout << std::setw(2) << std::setfill('0') << log[i].hour
        << ":00 -> " << log[i].temperature << std::endl;
        sum += log[i].temperature;
        if (log[i].temperature > log[maxIdx].temperature)
            maxIdx = i;
    }

    std::cout << "Average: " << sum / 5.0 << std::endl;
    std::cout << "Max temperature at hour "
    << log[maxIdx].hour << ": "
    << log[maxIdx].temperature << std::endl;
    return 0;
}
```

Explanation

An array of structs keeps related data together in a natural way — each element is a complete Reading with its own hour and temperature. Tracking maxIdx (the index of the current maximum) rather than copying the value lets you retrieve all fields of the winning entry at the end without a second search. std::setw and std::setfill('0') pad the hour to two digits for clean output.

Struct with Comparison



Topic: Structs / Comparison / Ternary

Solution Code

```
#include <iostream>
#include <string>

struct Student {
    std::string name;
    int grade;
};

int main() {
    Student s1 = {"Alice", 88};
    Student s2 = {"Bob", 95};
    Student top = (s1.grade > s2.grade) ? s1 : s2;
    std::cout << "Top student: " << top.name
    << " (" << top.grade << ")"
    << std::endl;
    return 0;
}
```

Explanation

A *struct* groups related data. Members are accessed with the dot operator (*s1.name*). The ternary operator *?:* is a compact *if/else* for choosing between two values — it evaluates the condition and returns one of the two expressions. Structs support assignment with *=* so *top* receives a full copy of the winning student.