2 – Exercises on booleans

Quick reference

Operator	Meaning	Example
`&&`	AND — true only if *both* sides are true	`a && b`
`||`	OR — true if *at least one* side is true	`a || b`
`!`	NOT — flips the value	`!a`
`==`	equal to	`a == b`
`!=`	not equal to	`a != b`
<, >, <=, >=	comparisons	`a < b`

In C++, `false` is stored as `0` and `true` as `1`. Printing a `bool` with `std::cout` will display `0` or `1` by default.

Problem 1 — True or false? (warm-up)

For each expression below, write **true** or **false**. The variables are:

Listing 1:

```
1  bool a = true;
2  bool b = false;
3  bool c = true;
```

(a) `a && b`

(b) `a || b`

(c) `!b`

(d) `!a || c`

(e) `a && b && c`

(f) `!a || !b || !c`

(g) `!(a && c)`

(h) `!(a || b) && c`

Problem 2 — Comparison operators

Let `volume = 75`, `pitch = 440`, `brightness = 200`. Evaluate each expression to **true** or **false**.

(a) `volume > 50`

(b) `pitch == 440`

(c) `brightness != 200`

(d) `volume >= 75 && pitch < 500`

(e) `brightness > 255 || volume == 75`

(f) `!(pitch > 400) || brightness <= 100`

(g) `volume == 75 && pitch != 440`

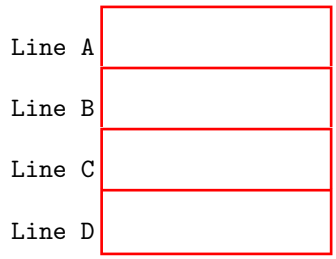
(h) `!(volume < 80 && brightness > 100)`

Problem 3 — Trace the code by hand

What does this program print? Write the output for each `std::cout` line.

Listing 2: Studio session

```cpp
#include <iostream>

int main() {
    bool is_recording = true;
    bool is_muted    = false;
    bool has_signal = true;

    bool can_hear = has_signal && !is_muted;
    bool on_air  = is_recording && can_hear;
    bool silent  = !has_signal || is_muted;

    std::cout << can_hear << std::endl; // line A
    std::cout << on_air << std::endl; // line B
    std::cout << silent << std::endl; // line C

    is_muted = true;
    can_hear = has_signal && !is_muted;

    std::cout << can_hear << std::endl; // line D

    return 0;
}
```



Problem 4 — Truth table

Fill in the complete truth table for the expression:

```
result = (a && !b) || (!a && b)
```

a	b	!b	a && !b	!a && b	result
false	false				
false	true				
true	false				
true	true				

This expression is an operation called **XOR**: it is an exclusionary OR.

Problem 5 — Spot the bug (image filter)

The function below is supposed to check whether a pixel is *valid* (each channel, `r`, `g` and `b` is in the range 0–255) **and** *bright* (average of channels above 128). It contains a logical bug. Find it and write the fix.

Listing 3: Buggy pixel validator

```
int main() {
    int r = 200, g = 180, b = 90;

    bool valid = (r >= 0 && r <= 255) ||
                 (g >= 0 && g <= 255) ||
                 (b >= 0 && b <= 255);

    int  avg    = (r + g + b) / 3;
    bool bright = avg > 128;

    bool ok = valid && bright;
    return ok;
}
```

Problem 6 — Write the code: concert gate

A concert has the following entry rules:

- The visitor must have a **ticket** (`has_ticket`).
- The visitor must **not** be on the banned list (`is_banned`).
- VIP visitors (`is_vip`) can enter through a **side door** even without a ticket, but still cannot enter if banned.

Write a complete `main` function that:

1. Declares four `bool` variables: `has_ticket`, `is_banned`, `is_vip`, and `can_enter`.
2. Assigns `has_ticket = false`, `is_banned = false`, `is_vip = true`.
3. Computes `can_enter` using a single boolean expression that captures all three rules above.
4. Returns `can_enter`.

Challenge: what combination of values makes `can_enter` false even for a VIP?

Problem 7 — Write the code: audio clipping detector

Prolonged exposure to sound levels exceeding 85 to 90 dB(A) puts hearing at risk. This is called the hearing danger threshold.

Write a complete `main` that:

1. Declares three `int` variables `s1`, `s2`, `s3` representing sound levels, and assigns them the values `-5`, `75`, and `120`.
2. For each level, declares a `bool` that is `true` when the sound level is dangerous for prolonged exposures. Name them `is_dangerous1`, `is_dangerous2`, `is_dangerous3`.
3. Declares a `bool` named `any_is_dangerous` that is `true` if *at least one* sound level is above the danger threshold.
4. Declares a `bool` named `all_ok` that is `true` only if *no* sound level is above the hearing danger threshold.
5. Returns `any_is_dangerous`.

Expected values:

```
is_dangerous1

is_dangerous2

is_dangerous3

any_is_dangerous

all_ok
```