4 – Exercises on Switch-Case and Enums

Université Jean Monnet – Saint-Étienne 2026/2027

Quick reference

Listing 1:

```
enum class Season { Spring, Summer, Autumn, Winter };

Season s = Season::Summer;

switch (s) {
    case Season::Spring: std::cout << "spring"; break;
    case Season::Summer: std::cout << "summer"; break;
    default:             std::cout << "other"; break;
}
```

Key rules to remember:

- If `case` does not end with `break`, the execution *falls through* to the next case.
- `default` catches any value not listed in a `case`.
- With `enum class`, you must always prefix the value with the enum name: `Season::Summer`, not just `Summer`.

Part A — Trace by hand

Problem 1 — What does this print?

Listing 2: Instrument family

```
#include <iostream>

enum class Instrument { Piano, Guitar, Drums, Violin };

int main() {
    Instrument i = Instrument::Guitar;
    switch (i) {
        case Instrument::Piano:
            std::cout << "keys" << std::endl;
            break;
        case Instrument::Guitar:
            std::cout << "strings" << std::endl;
            break;
        case Instrument::Drums:
            std::cout << "percussion" << std::endl;
            break;
```

```
        case Instrument::Violin:
            std::cout << "strings" << std::endl;
            break;
    }

    return 0;
}
```

Output:

What would be printed for each value below?

`Instrument::Piano` ⇒

`Instrument::Drums` ⇒

`Instrument::Violin` ⇒

Problem 2 — Spot the fallthrough

What does this program print?

Listing 3: Filter type

```
#include <iostream>

enum class Filter { Blur, Sharpen, Greyscale, Invert };

int main() {
    Filter f = Filter::Sharpen;

    switch (f) {
        case Filter::Blur:
            std::cout << "applying blur"   << std::endl;
            break;
        case Filter::Sharpen:
            std::cout << "applying sharpen" << std::endl;
        case Filter::Greyscale:
            std::cout << "applying greyscale" << std::endl;
            break;
        case Filter::Invert:
            std::cout << "applying invert" << std::endl;
            break;
    }

    return 0;
}
```

Output (list every line printed):

Line 1:

Line 2:

Problem 3 — Default branch

What does this print?

Listing 4: Playback state

```
#include <iostream>

enum class Playback { Playing, Paused, Stopped };

int main() {
    Playback state = Playback::Paused;
    int      count = 0;

    switch (state) {
        case Playback::Playing:
            count = count + 10;
            break;
        case Playback::Stopped:
            count = count + 1;
            break;
        default:
            count = count + 5;
            break;
    }

    std::cout << count << std::endl;
    return 0;
}
```

Output:

What would be printed for `state = Playback::Playing`?

What would be printed for `state = Playback::Stopped`?

Note: `Playback::Paused` is not listed as a `case` — which branch handles it?

Problem 4 — Shared cases

What does this print?

Listing 5: Warm or cool colour?

```cpp
#include <iostream>

enum class Colour { Red, Orange, Yellow, Green, Blue, Violet };

int main() {
    Colour c = Colour::Orange;

    switch (c) {
        case Colour::Red:
        case Colour::Orange:
        case Colour::Yellow:
            std::cout << "warm" << std::endl;
            break;
        case Colour::Green:
            std::cout << "neutral" << std::endl;
            break;
        case Colour::Blue:
        case Colour::Violet:
            std::cout << "cool" << std::endl;
            break;
    }

    return 0;
}
```

Output:

What would be printed for each value below?

`Colour::Red` ⇒

`Colour::Blue` ⇒

`Colour::Green` ⇒

`Colour::Violet` ⇒

What technique does this exercise demonstrate? How does it work without `break` between shared cases?

Problem 5 — Trace with a variable

What value does `main` return?

Listing 6: Genre score

```cpp
enum class Genre { Pop, Rock, Jazz, Classical };

int main() {
    Genre g    = Genre::Jazz;
    int   score = 0;

    switch (g) {
        case Genre::Pop:    score = 1; break;
        case Genre::Rock:   score = 2; break;
        case Genre::Jazz:   score = 10; break;
        case Genre::Classical:score = 7; break;
    }

    score = score * 3;
    return score;
}
```

Return value:

What would be returned for `g = Genre::Classical`?

Part B — Spot the bug

Problem 6 — Three bugs to find

The program below contains **three separate bugs**. Identify each one and describe the fix.

Listing 7: Buggy layer type

```cpp
#include <iostream>

enum class Layer { Background, Midground, Foreground };

int main() {
    Layer l = Foreground;                // bug 1

    switch (l) {
        case Layer::Background:
            std::cout << "back" << std::endl;
        case Layer::Midground:           // bug 2
            std::cout << "mid" << std::endl;
            break;
        case Layer::Foreground:
            std::cout << "front" << std::endl;
            break;
        case Layer::Overlay:             // bug 3
            std::cout << "overlay" << std::endl;
```

```
19              break;
20      }
21
22      return 0;
23  }
```

Bug 1 (line 7):

Bug 2 (line 10):

Bug 3 (line 16):

Part C — Write the code

Problem 7 — Band role

Define an `enum class` called `Role` with four values: `Singer`, `Guitarist`, `Bassist`, `Drummer`.

Write a `main` that:

1. Declares a variable of type `Role` set to `Role::Bassist`.
2. Uses a `switch` to print one sentence describing what that role does in the band.
3. Has a `default` branch that prints `"unknown role"`.

Expected output for `Role::Bassist`: something like `"holds down the low end"` (your wording).

Problem 8 — Image channel (harder)

Define an `enum class` called `Channel` with values `Red`, `Green`, `Blue`, `Alpha`.

Write a `main` that:

1. Declares `Channel c = Channel::Alpha` and `int value = 128`.
2. Uses a `switch` on `c` to add a different bonus to `value`: `Red` → `+10`, `Green` → `+20`, `Blue` → `+5`, `Alpha` → `+0` (no change).
3. After the switch, checks with an `if` whether `value` is above 255, and if so sets it to 255 (clamping).
4. Returns `value`.

Expected return value for `Channel::Alpha`:

Expected return value for `Channel::Green`:

Which channel(s) would cause clamping if `value` started at 250?