14 – Exercises on Classes

Université Jean Monnet – Saint-Étienne 2026/2027

Quick reference

```
struct Point {          // all members public by default
    float x;
    float y;
    float distance() { return x * x + y * y; } // member function
};

class Volume {          // all members private by default
public:
    Volume(int v) : value(v) {} // constructor with initialiser list

    int  get()       const { return value; }
    void set(int v)       { value = clamp(v); }

private:
    int value;
    int clamp(int v) { return v < 0 ? 0 : (v > 100 ? 100 : v); }
};

Volume vol(80);         // construction
vol.set(120);           // calls set(), which calls private clamp()
int v = vol.get();      // v == 100
```

Key rules:

- `struct` defaults to `public`; `class` defaults to `private`.
- A **constructor** has the same name as the class, no return type, and runs whenever an object is created.
- The **initialiser list** (`: member(value)`) initialises members before the constructor body runs — prefer it over assignment inside the body.
- `const` after a member function signature means the function does not modify the object.
- **Abstraction**: callers use the public interface and do not need to know how the private implementation works.

Part A — Structs

Problem 1 — Trace the output

What does this program print?

A simple struct

```cpp
#include <iostream>

struct Track {
    std::string title;
    int         duration;
    int         bpm;

    void print() const {
        std::cout << title << " (" << duration << "s, "
                  << bpm << " bpm)" << std::endl;
    }

    bool is_long() const { return duration > 240; }
};

int main() {
    Track a;
    a.title   = "So What";
    a.duration = 340;
    a.bpm     = 130;

    Track b = {"Blue in Green", 185, 90};

    a.print();                                  // line A
    b.print();                                  // line B
    std::cout << a.is_long() << std::endl;      // line C
    std::cout << b.is_long() << std::endl;      // line D

    a.duration = b.duration;
    std::cout << a.is_long() << std::endl;      // line E

    return 0;
}
```

Line A:

Line B:

Line C:

Line D:

Line E:

Problem 2 — Adding a member function

The struct below is missing a member function. Add it.

Pixel struct

```
struct Pixel {
    int r, g, b;

    // Add a member function:
    // int brightness() const
    // that returns the average of r, g, b (integer division).
};
```

Then write the `main` below and predict its output before running it.

```
int main() {
    Pixel p = {120, 60, 30};
    std::cout << p.brightness() << std::endl;
    p.r = 255;
    std::cout << p.brightness() << std::endl;
    return 0;
}
```

Expected output:

Line 1: Line 2:

Problem 3 — Spot the bug

Each snippet has one bug related to structs. Identify and fix it.

Three struct bugs

```
// (1) Should print the duration of the track
struct Track { std::string title; int duration; };
Track t;
t.title   = "Echoes";
t.duration = 340;
std::cout << Track.duration << std::endl; // bug

// (2) What gets printed ?
struct Pixel { int r, g, b; };
Pixel p;
std::cout << p.r << std::endl;           // bug

// (3) Does this compile?
struct Counter { int count;
    void increment() const { count = count + 1; } // bug
};
```

Bug 1:

Fix 1:

Bug 2:

Bug 3:

Fix 3:

Part B — Classes, access control and constructors

Problem 1 — Public vs. private

For each line marked `// (?)` below, write whether it **compiles** or causes a **compile error**, and why.

Access control

```
class AudioClip {
public:
    AudioClip(int dur, float vol) : duration(dur), volume(vol) {}

    int   get_duration() const { return duration; }
    void  set_volume(float v) { volume = clamp(v); }
    float get_volume() const { return volume; }

```

```
private:
    int   duration;
    float volume;
    float clamp(float v) { return v<0.f ? 0.f : (v>1.f ? 1.f : v); }
};

int main() {
    AudioClip clip(180, 0.8f);

    std::cout << clip.get_duration() << std::endl; // (a)
    std::cout << clip.duration    << std::endl; // (b)
    clip.set_volume(1.5f);                      // (c)
    std::cout << clip.get_volume() << std::endl; // (d)
    std::cout << clip.clamp(0.5f) << std::endl; // (e)
    clip.volume = 0.2f;                         // (f)
    return 0;
}
```

(a) because:

(b) because:

(c) because:

(d) because:

(e) because:

(f) because:

After line (c), what does `clip.get_volume()` return, and why?

Problem 2 — Trace the constructor

What does this program print?

Constructor and initialiser list

```
#include <iostream>

class Metronome {
public:
    Metronome(int bpm, int beats_per_bar)
        : bpm(bpm), beats_per_bar(beats_per_bar), tick(0) {
        std::cout << "created: " << bpm << " bpm" << std::endl;
    }
```

```
9
10      void advance() {
11          tick = tick + 1;
12          if (tick % beats_per_bar == 0)
13              std::cout << "bar!" << std::endl;
14      }
15      int get_tick() const { return tick; }
16
17  private:
18      int bpm;
19      int beats_per_bar;
20      int tick;
21  };
22
23  int main() {
24      Metronome m(120, 4);        // line A
25      m.advance();                // line B
26      m.advance();                // line C
27      m.advance();                // line D
28      m.advance();                // line E
29      std::cout << m.get_tick() << std::endl; // line F
30
31      Metronome m2(90, 3);        // line G
32      m2.advance();
33      m2.advance();
34      m2.advance();               // line H
35
36      return 0;
37  }
```

List every line of output in order:

Line A:

Line B:

Line C:

Line D:

Line E:

Line F:

Line G:

Line H:

The constructor parameter is also named `bpm`, same as the member. Inside the initialiser list `: bpm(bpm)`, which `bpm` is which?

Problem 3 — Spot the bug

Each class below has one bug. Identify and fix it.

Three class bugs

```
1  // 1
2  class Counter {
3  public:
4      Counter(int start) : start(start) {}
5      void increment()  { count = count + 1; }
6      int  get() const  { return count; }
7  private:
8      int start;
9      int count;
10 };
11
12 // 2
13 class Note {
14 public:
15     Note(int v) : value(v) {}
16     int get() const { value = value + 1; return value; }
17 private:
18     int value;
19 };
20
21 // 3
22 class Volume {
23 public:
24     Volume(int v) : level(v) {}
25     int get() const { return level; }
26 private:
27     int level;
28 };
29 Volume vol(80);
30 vol.level = 50;
```

Bug 1:

Fix 1:

Bug 2:

Fix 2:

Bug 3:

Fix 3:

Part C — Abstraction

Problem 1 — What is hidden?

Read this class declaration and answer the questions below.

Colour class

```
class Colour {
public:
    Colour(int r, int g, int b);

    int  red()        const;
    int  green()      const;
    int  blue()       const;
    int  brightness() const;
    void darken(int amount);
    void lighten(int amount);

private:
    int r, g, b;
    int clamp(int v) const;
};
```

Answer in plain sentences:

What can a caller access in a `Colour` object?

What is hidden from the caller?

If you change the body of `clamp`, do callers need to change their code? Why?

Problem 2 — Write the code: `class Fader`

Design and implement a class `Fader` that models a mixing desk fader (a volume control).

Requirements:

- The fader holds a level between 0 and 100 (integers).
- The constructor takes an initial level and clamps it immediately.
- `int get_level()const` — returns the current level.
- `void set_level(int v)` — sets the level, clamping to $[0, 100]$.
- `void nudge(int amount)` — adds `amount` to the level (positive or negative), clamping the result.
- `bool is_muted()const` — returns `true` when the level is 0.
- `void print()const` — prints `"fader: <level>"` followed by `" [muted]"` if the fader is muted.
- The level variable must be `private`.
- The clamping logic must be factored into a single `private` helper.

Expected output for the following `main`:

```
int main() {
    Fader f(80);
    f.print();
    f.nudge(30);
    f.print();
    f.set_level(0);
    f.print();
    f.nudge(-10);
    f.print();
    Fader g(150);
    g.print();
    return 0;
}
```

```
fader: 80
fader: 100
fader: 0 [muted]
fader: 0 [muted]
fader: 100
```

Problem 3 — Write the code: `class Setlist`

Design and implement a class `Setlist` that manages an list of song titles.

Requirements:

- Internally stores titles in a `std::vector<std::string>` (private).

- Default constructor creates an empty setlist.
- `void add(const std::string& title)` — appends a title if it is not already present; if it is, prints `"duplicate: <title>"` and does not add it.
- `int size()const` — returns the number of songs.
- `bool contains(const std::string& title)const` — returns `true` if the title is in the list. Use `std::find`.
- `void print()const` — prints each title on its own line, prefixed by its 1-based position.
- `void clear()` — removes all songs.

Expected output for the following `main`:

```
int main() {
    Setlist s;
    s.add("So What");
    s.add("Freddie Freeloader");
    s.add("So What");
    s.add("All Blues");
    s.print();
    std::cout << s.size() << std::endl;
    std::cout << s.contains("All Blues") << std::endl;
    std::cout << s.contains("Giant Steps") << std::endl;
    s.clear();
    std::cout << s.size() << std::endl;
    return 0;
}
```

```
duplicate: So What
1: So What
2: Freddie Freeloader
3: All Blues
3
1
0
0
```