Assignment 1

Université Jean Monnet – Saint-Étienne 2026/2027

Each exercise is independent. Read the full statement before writing any code. Your program must compile and produce exactly the expected output described.

Exercise 1 — Concert night

You can use this godbolt session to complete this exercise : `https://godbolt.org/z/5Pjnqfzf9`.

A small venue hosts concerts. Each evening, a single artist performs one set. You will write a program that manages one such evening.

Step 1 — The artist and their genre.

Define an `enum class` called `Genre` with the following values: `Rock`, `Jazz`, `Pop`, `Classical`.

In `main`, declare a variable of type `Genre` and assign it `Genre::Jazz`.

Use a `switch` to set an `int` variable `base_price` according to the genre:

Genre	Base ticket price
`Rock`	25
`Jazz`	30
`Pop`	20
`Classical`	35

Step 2 — Audience and revenue.

Declare the following variables:

- `const int capacity = 120` — total seats in the venue.
- `const int tickets_sold = 97` — seats actually filled.
- `int revenue` — to be computed as `tickets_sold * base_price`.

Step 3 — Is the show sold out?

Declare a `bool` called `sold_out` that is `true` when the amount of `tickets_sold` exceeds the value of `capacity`.

Print one line depending on its value:

- `true` → print `"Full house!"`
- `false` → print `"seats available"`

Table 1: Revenue rating

Condition	Print
`revenue >= 3000`	`"great evening"`
`revenue >= 2000`	`"decent evening"`
otherwise	`"rough evening"`

Step 4 — Revenue report.

Print the revenue on a line of its own.

Then print a rating based on the revenue as described in TABLE 1:

Expected output (for the values given above):

Listing 1:

```
seats available
2910
decent evening
```

Step 5 — Profitability study.

You want to know which music genre generates the biggest profit. Change your program variables's values according to TABLE 2, and fill-in the blanks.

Table 2: Concerts numbers recap'

Genre	`tickets_sold`	`revenue`
`Rock`	120	
`Pop`	115	
`Classical`	86	

Exercise 2 — Image histogram

This exercise **must** be completed in the following Godbolt session: `https://godbolt.org/z/hP1KGscj7`.

A greyscale image is a grid of pixels. Each pixel holds an integer value between 0 (black) and 255 (white). You will write a program that analyses a small image represented as a sequence of pixel values.

Step 1 — Pixel categories.

Define an `enum class` called `Shade` with three values: `Dark`, `Mid`, `Bright`.

A pixel is of a certain shade – `Dark`, `Mid` or `Bright` – when it has a certain value. The shades' values are defined as follows:

- `Dark` — value strictly below 85
- `Mid` — value from 85 to 169 inclusive
- `Bright` — value 170 or above

Step 2 — The image. (Expected: 1 line of output)

In the remainder of this exercise, you can use the following variable : `PIXEL_COUNT`. **You do not need to declare it**.

We consider an image that has `PIXEL_COUNT` pixels. If you declare a variable of type `int`, for example `int pixel;` , you can write the following: `pixel = get_next_pixel();` . This will store in `pixel` the value of the next pixel in the image.

For example, if the image has three pixels, then `PIXEL_COUNT=3`. Let's say those pixels have the following values: `25, 42, 211`. In that case, the following code:

Listing 2:

```
int pixel = get_next_pixel();
std::cout << "Pixel 1: " << pixel << std::endl;
pixel = get_next_pixel();
std::cout << "Pixel 2: " << pixel << std::endl;
pixel = get_next_pixel();
std::cout << "Pixel 3: " << pixel << std::endl;
```

will output :

Listing 3:

```
Pixel 1: 25
Pixel 2: 42
Pixel 3: 211
```

When you have written `pixel = get_next_pixel();` for `PIXEL_COUNT` times, writting `pixel = get_next_pixel();` another time will give the value of the first pixel again, and so on.

Do the following: In the provided godbolt session, `PIXEL_COUNT=1000`. Using a loop of your choice and `get_next_pixel()`, output the value of each pixel in the image. Print them all on the same line, each value separated by a space like so:

Listing 4:

```
11 43 201 242 78 156
```

Step 3 — Classify and count.

Declare three `int` counters, all initialised to 0: `dark_count`, `mid_count`, `bright_count`.

Using a loop and a sequence of `if` / `else if` / `else` blocks, increment the corresponding counter according to each pixel value.

Step 4 — Summary. (Expected: 4 lines of output)

Print the three counts, each on its own line, in this format:

Listing 5:

```
dark: N
mid: N
bright: N
```

Then declare a `bool` called `balanced` that is `true` when no single category holds more than 500 pixels.

Print `"balanced"` if `balanced` is true, and `"unbalanced"` otherwise.

Step 5 — Brightest pixel. (Expected: 1 line of output)

Using a `for` loop, find the largest value amongst the image pixels.

Print `"brightest: N"` where `N` is the maximum value found.

Step 6 — Shade of the brightest. (Expected: 1 line of output)

Declare a variable of type `Shade` and assign it the correct shade for the brightest pixel found in step 5.

Use a `switch` to print one of: `"shade: dark"`, `"shade: mid"`, `"shade: bright"`.

You can check your answers by copy-pasting your program output into this godbolt session : `https://godbolt.org/z/n7GPsWc1b`. The exercise expects 7 lines of output.

Important: If you couldn't complete a step, leave blank line(s) for the output corresponding to that step. For example, if you couldn't complete step 5, write:

Listing 6:

```
1 2 3 4 5
dark: 1
mid: 2
bright: 3
balanced

shade: mid
```