ASSIGNMENT 3

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.

For references on handling `main`'s argument, see the Quick Reference of Lecture 2's exercises on the `main` function.

Exercise 1 — Setlist manager

You can complete this exercise by starting in this session: `https://godbolt.org/z/9vG3od7va`. Don't forget to export your session link once you're done.

You will write a command-line program that manages a concert setlist stored as a `std::vector`.

Invocation:

```
./setlist <command> [args...]
```

Step 1 — Global state and types

Declare a global `std::vector<int>` called `setlist`. Each element represents the duration (in seconds) of one song.

Define an `enum class Tempo` with values `Slow`, `Medium`, `Fast`.

Step 2 — Helper functions

Write the following functions.

`void add_song(int duration)` — appends `duration` to `setlist`. If `duration <= 0`, print `"invalid duration"` and do not append.

`int total_duration()` — returns the sum of all durations in `setlist`.

`int longest()` — returns the longest duration in `setlist`, or `0` if the setlist is empty.

`Tempo classify(int duration)` — returns:

Condition	Tempo
`duration < 180`	`Slow`
`duration < 300`	`Medium`
otherwise	`Fast`

`void print_tempo(Tempo t)` — uses a `switch` to print `"slow"`, `"medium"`, or `"fast"` followed by a newline.

`void print_setlist()` — prints each song on its own line as `"song N: <duration>s"` (1-indexed), then prints the total duration on a final line as `"total: <duration>s"`.

Step 3 — Command dispatch in `main`

`main` reads `argv[1]` and dispatches as follows. Use `if` / `else if` / `else`.

Command `add` — takes one extra argument: a duration. Calls `add_song`, then prints the current size of the setlist: `"setlist: N songs"`.

Command `build` — takes a variable number of extra arguments, each a duration. Calls `add_song` for each one, then calls `print_setlist()`.

Command `analyse` — takes a variable number of extra arguments (durations). Builds the setlist, then prints:

- the total duration,
- the longest song,
- the tempo class of the longest song (using `classify` and `print_tempo`).

If no command is given or the command is unknown, print:

```
usage: ./setlist <add|build|analyse> [durations...]
```

and return `1`.

Step 4 — Expected outputs

Invokation example

```
./setlist add 240
```

Expected output

```
setlist: 1 songs
```

Invokation example

```
./setlist build 210 185 340 95 270
```

Expected output

```
song 1: 210s
song 2: 185s
song 3: 340s
song 4: 95s
song 5: 270s
total: 1100s
```

Invokation example

```
./setlist analyse 210 185 340 95 270
```

Expected output

```
total: 1100s
longest: 340s
fast
```

Invokation example

```
./setlist add -5
```

Expected output

```
invalid duration
setlist: 0 songs
```

Exercise 2 — RGB image processor

This exercise has to be done in `https://godbolt.org/z/48MPbbv5K`.

You will write a program that processes small RGB images as 3D arrays of shape: 3 $\times$ rows $\times$ columns. The color channels are in the order: red, green, blue.

The images are stored and accessible in the following variables:

Variable name	Variable type	Rows	Columns
`cat`	`std::array<std::array<std::array<int, cols>, rows>, 3>`	47	32
`dog`	`std::array<std::array<std::array<int, cols>, rows>, 3>`	47	32
`heart`	`std::array<std::array<std::array<int, cols>, rows>, 3>`	47	32

These are 3D arrays. The first index corresponds to accessing the underlying 2D color channel. Then, you index the row, and then the column. Look at the following examples:

Examples of indexing 3D arrays

```
1 cat[0]; // This is the red channel of the cat image. It has type std::array<std::array
      <int, cols>, rows>
2 dog[1][12][14] = 0; // sets the green channel of pixel at (12, 14) to 0
```

To simplify the code you can write `Image` instead of `std::array<std::array<std::array<int, cols>, height>, 3>`.

You have access to the variables `cols` and `rows` that contain the amount of columns and rows respectively in each image.

Step 1 — Helper functions

Write the following functions, each taking a `const Image&` (or a plain `Image` for `apply_offset`).

`void print_image(const Image& img)` — prints each pixel as `(R,G,B)` next to each other, one row per line. An example would look like:

```
(255,0,0)(0,255,0)(0,0,255)
(255,0,0)(0,255,0)(0,0,255)
```

`int channel_sum(const Image& img, int channel)` — returns the sum of the given channel (0 = red, 1 = green, 2 = blue) across all pixels.

`int brightness(const Image& img, int row, int col)` — returns the average of the three channels of pixel `(row, col)` (integer division).

`int brightest(const Image& img)` — returns the value of the brightest pixel.

`Image apply_offset(Image img, int offset, int channel)` — adds `offset` to channel `channel` of every pixel, clamping each result to $[0, 255]$, and returns the modified image. Note: the image is passed **by value** so the original is not modified.

`Image apply_offset(Image img, int offset)` — adds `offset` to every channel of every pixel, clamping each result to $[0, 255]$, and returns the modified image. Note: the image is passed **by value** so the original is not modified.

Step 2 The `process` function

Write a function `void process(const Image& image, const int offset);` that :

- Prints the value of the brightest pixel of `image`
- Applies the offset `offset` to the given `image`
- Prints the resulting image's pixels with `print_image`

Step 3 — The `main` function

The program will take exactly two arguments:

- The first is one of the following strings: `cat`, `dog`, or `heart`;
- The second is an integer that is at most 255.

Write the code in `main` that checks the program's input argument. If `argc != 3`, the program should output: `usage: ./main <cat|dog|heart> <offset>`.

Then, call `process` with the image whose name is the first argument given to the program and with the offset given as second argument to the program.

Step 4 — Check your result

You can visualize your modified picture by copy-pasting the output of `print_image` in the left text pad of `https://grybouilli.github.io/extra/img_visualizer.html`