Assignment 6

Université Jean Monnet – Saint-Étienne 2026/2027

Read the full assignment before writing any code. All expected outputs can be verified by running your program.

Part A — Provided structures

The following declarations are given. Copy them into your project. The goal of this part will be to implement their functionalities.

Listing 1: Provided declarations — do not modify

```cpp
#include <string>
#include <vector>

class Artist {
    std::vector<std::string> _members;
public:
    std::string name;

    Artist(const std::string name,
           const std::vector<std::string> members = {});

    bool has_members() const;
};

struct Song {
    std::string name;
    Artist      author;
    int         duration;  // in seconds
    int         bpm;
};

struct Image {
    std::string name;
    Artist      author;
    int         width;     // in pixels
    int         height;    // in pixels
};
```

Exercise 1 — Implement `Artist`

Implement the two members declared in `Artist`.

- `Artist(const std::string name, const std::vector<std::string> members)`

  Stores `name` in the public member `name` and `members` in `_members`.

- `bool has_members()const`

  Returns `true` if `_members` is non-empty, `false` otherwise.

Verify with the following snippet before moving on:

```
Artist solo("Miles Davis");
Artist band("Pink Floyd", {"Waters", "Gilmour", "Mason", "Wright"});

std::cout << solo.name     << std::endl; // Miles Davis
std::cout << solo.has_members() << std::endl; // 0
std::cout << band.name     << std::endl; // Pink Floyd
std::cout << band.has_members() << std::endl; // 1
```

Exercise 2 — Free functions on `Song` and `Image`

Write the following plain (non-template) free functions.

- `void print_song(const Song& s)`

  Prints one line:

```
"<name>" by <author.name> (<duration>s, <bpm> bpm)
```

- `void print_image(const Image& img)`

  Prints one line:

```
"<name>" by <author.name> (<width>x<height>)
```

- `bool same_author(const Song& s, const Image& img)`

  Returns `true` if both share the same `author.name`.

Verify:

```
Artist miles("Miles Davis");
Artist floyd("Pink Floyd", {"Waters", "Gilmour", "Mason", "Wright"});

Song s1 = {"So What",     miles, 562, 136};
Song s2 = {"Breathe",     floyd, 163, 60};
Image i1 = {"Kind of Blue", miles, 1200, 1200};
Image i2 = {"The Wall",   floyd, 800,  800};
```

```
print_song(s1);    // "So What" by Miles Davis (562s, 136 bpm)
print_song(s2);    // "Breathe" by Pink Floyd (163s, 60 bpm)
print_image(i1);   // "Kind of Blue" by Miles Davis (1200x1200)
print_image(i2);   // "The Wall" by Pink Floyd (800x800)

std::cout << same_author(s1, i1) << std::endl; // 1
std::cout << same_author(s1, i2) << std::endl; // 0
```

Exercise 3 — Generic functions

Write the following function templates. They must work on both `Song` and `Image` (and any other type that has a `name` field and an `author` member with a `name` field and `has_members()` method).

(a) `template<typename T> bool is_by_band(const T& item)`

Returns `true` if `item.author.has_members()`.

(b) `template<typename T> void print_name_and_author(const T& item)`

Prints one line:

```
"<name>" -- <author.name>
```

(c) `template<typename T> int count_by_band(const std::vector<T>& items)`

Returns the number of items whose author is a band, i.e `item.author.has_members()` returns true.

(d) `template<typename T> std::vector<T> filter_by_author(const std::vector<T>& items, const std::string& author_name)`

Returns a new vector containing only the items whose `author.name == author_name`.

Verify with the following snippet:

```
std::vector<Song> songs = {s1, s2, Song{"Kind of Blue Theme", miles, 340, 120}};
std::vector<Image> images = {i1, i2, Image{"Mystery", miles, 600, 600}};

std::cout << get_name(s1)           << std::endl; // So What
std::cout << is_by_band(s1)         << std::endl; // 0
std::cout << is_by_band(s2)         << std::endl; // 1
print_name_and_author(i1);                        // "Kind of Blue" -- Miles Davis
std::cout << count_by_band(songs)   << std::endl; // 1
std::cout << count_by_band(images)  << std::endl; // 1

std::vector<Song> miles_songs = filter_by_author(songs, "Miles Davis");
for (int i = 0; i < miles_songs.size(); i = i + 1)
    print_name_and_author(miles_songs[i]);
// "So What" -- Miles Davis
// "Kind of Blue Theme" -- Miles Davis
```

Part B — `class Playlist`

Write a class `Playlist` that manages a collection of `Song` objects.

Private members:

- `std::string _name`
- `std::vector<Song> _songs`

Constructor: `Playlist(const std::string& name)` — stores the name, starts with an empty song list.

Public methods:

- `void add(const Song& s)` — appends the song to `_songs`.
- `int size()const` — returns the number of songs.
- `int total_duration()const` — returns the sum of all `duration` fields.
- `const Song& longest()const` — returns the song with the largest `duration`. Assume the playlist is non-empty.
- `std::vector<Song> by_author(const std::string& author_name)const` — returns all songs by that author. Use your `filter_by_author` template.
- `void print()const` — prints the playlist name on the first line, then each song using `print_song`, prefixed by its 1-based index.

Verify with the following `main` fragment:

```
Playlist pl("Evening set");
pl.add(Song{"So What",         miles, 562, 136});
pl.add(Song{"Breathe",         floyd, 163, 60});
pl.add(Song{"Kind of Blue Theme", miles, 340, 120});
pl.add(Song{"Time",            floyd, 413, 72});
pl.add(Song{"All Blues",       miles, 698, 108});

pl.print();
std::cout << "total: " << pl.total_duration() << "s" << std::endl;
std::cout << "longest: " << pl.longest().name << std::endl;

std::vector<Song> floyd_songs = pl.by_author("Pink Floyd");
std::cout << "Pink Floyd tracks: " << floyd_songs.size() << std::endl;
```

Expected output:

```
Evening set
```

```
1: "So What" by Miles Davis (562s, 136 bpm)
2: "Breathe" by Pink Floyd (163s, 60 bpm)
3: "Kind of Blue Theme" by Miles Davis (340s, 120 bpm)
4: "Time" by Pink Floyd (413s, 72 bpm)
5: "All Blues" by Miles Davis (698s, 108 bpm)
total: 2176s
longest: All Blues
Pink Floyd tracks: 2
```

Part C — `class Album`

Write a class `Album` that manages a collection of `Image` objects. It mirrors `Playlist` for images.

Private members:

- `std::string _name`
- `std::vector<Image> _images`

Constructor: `Album(const std::string& name)`.

Public methods:

- `void add(const Image& img)` — appends the image.
- `int size()const` — returns the amount of images in the album.
- `int total_pixels()const` — returns sum of `width * height` for every image.
- `const Image& largest()const` — returns the image with the largest `width * height`. Assume non-empty.
- `std::vector<Image> by_author(const std::string& author_name)const` — returns all images by that author. Use `filter_by_author`.
- `void print()const` — prints the album name, then each image using `print_image`, 1-based index.

Verify:

```
Album album("Sound and Vision");
album.add(Image{"Kind of Blue", miles, 1200, 1200});
album.add(Image{"The Wall",   floyd, 800,  800});
album.add(Image{"Mystery", miles, 600, 600});
album.add(Image{"Wish You Were", floyd, 1000, 1000});

album.print();
std::cout << "total pixels: " << album.total_pixels() << std::endl;
```

```
std::cout << "largest: "   << album.largest().name << std::endl;

std::vector<Image> miles_images = album.by_author("Miles Davis");
std::cout << "Miles Davis images: " << miles_images.size() << std::endl;
```

Expected output:

```
Sound and Vision
1: "Kind of Blue" by Miles Davis (1200x1200)
2: "The Wall" by Pink Floyd (800x800)
3: "Mystery" by Miles Davis (600x600)
4: "Wish You Were" by Pink Floyd (1000x1000)
total pixels: 3800000
largest: Kind of Blue
Miles Davis images: 2
```

Part D — putting it all together

Write a single `main` that:

1. Constructs the `Playlist` and `Album` from part B and C (same data).
2. Uses `count_by_band` on the playlist's songs and the album's images (you will need to write a function to expose the internal vector in both classes) and prints:

   ```
   band songs: 2
   band images: 2
   ```

3. Finds all items (songs and images alike) by `"Miles Davis"` using `filter_by_author` and prints each one with `print_name_and_author`:

   ```
   Miles Davis works:
   "So What" -- Miles Davis
   "Kind of Blue Theme" -- Miles Davis
   "All Blues" -- Miles Davis
   "Kind of Blue" -- Miles Davis
   "Mystery" -- Miles Davis
   ```

4. Takes two specific items — `Song{"So What", miles, 562, 136}` and `Image{"Kind of Blue", miles, 1200, 1200}` — and uses `same_author` and `is_by_band` to print:

   ```
   same author: 1
   So What is by a band: 0
   Kind of Blue is by a band: 0
   ```

For step 2, you may add a `const std::vector<Song>& get_songs()const` and `const std::vector<Image>& get_images()const` accessor to `Playlist` and `Album` respectively.