8 – Exercises on Matrices

Université Jean Monnet – Saint-Étienne 2026/2027

Quick reference

```cpp
#include <array>
#include <vector>

// std::array: fixed size, known at compile time
std::array<int, 4> notes = {60, 62, 64, 65};
notes[2]     = 70;         // access by index
notes.size() == 4;         // always 4

// std::vector: dynamic size, can grow at runtime
std::vector<int> track = {80, 90, 70};
track.push_back(100);      // now has 4 elements
track.size() == 4;
track[0]     == 80;

// 2D array (fixed rows and columns)
std::array<std::array<int, 3>, 2> grid = {{{10, 20, 30},
                                           {40, 50, 60}}};
grid[1][2] == 60;          // row 1, column 2

// 2D vector (dynamic rows and columns)
std::vector<std::vector<int>> image;
image.push_back({255, 0, 128});
image.push_back({64, 32, 16});
image[0][1] == 0;

// Nested loop over a 2D container
for (int row = 0; row < image.size(); row = row + 1) {
    for (int col = 0; col < image[row].size(); col = col + 1) {
        // use image[row][col]
    }
}
```

Part A — Matrix shape and indexing

In this part, indexing is done with numbers, and with the convention (row number, column number). Same goes for the shape.

Problem 1 — Shaping

Fill in Table 1 according to the following matrices.

Matrix A

42	7
10	99
5	1

Matrix B

Name
User
Admin
Guest

Matrix C

0.33	4.20	9.88	1.00	5.55

Matrix D

Red	Blue	Green
Pink	Cyan	Lime

Matrix E

10	5
20	8

Table 1

Matrix	Number of rows	Number of columns	Shape
A			
B			
C			
D			
E			

Problem 2 — Indexing

Refering to the matrices of the previous exercise, answer the following questions:

1. Element at index $(2, 2)$ of matrix A:
2. Element at index $(3, 1)$ of matrix B:
3. Index of value 5 in matrix A:
4. Index of value 1.00 in matrix C:
5. Element at index $(2, 3)$ of matrix D:
6. Element at index $(2, 1)$ of matrix E:
7. Index of value Cyan in matrix D:
8. Index of value 8 in matrix E:

Part B Manipulating spreadsheets

Problem 1 — Image channel manipulation

Open the file `landscape.xlsx` with o-spreadsheet.

- Go to the sheet “red”
- Set the value of A1 to 0
- Select and drag the bottom right corner of the A1 cell until the cell AG1
- While keeping the complete line selected, drag the bottom right corner of AG1 until AG33
- You have now filled the red channel of the landscape image with 0s
- Save the spreadsheet as a new .xlsx file
- Use the website image_exporter to export the .xlsx file to a .png file as illustrated on Figure 1
- Reproduce the experience for the channel green and the channel blue.

Figure 1: Using `image_exporter`

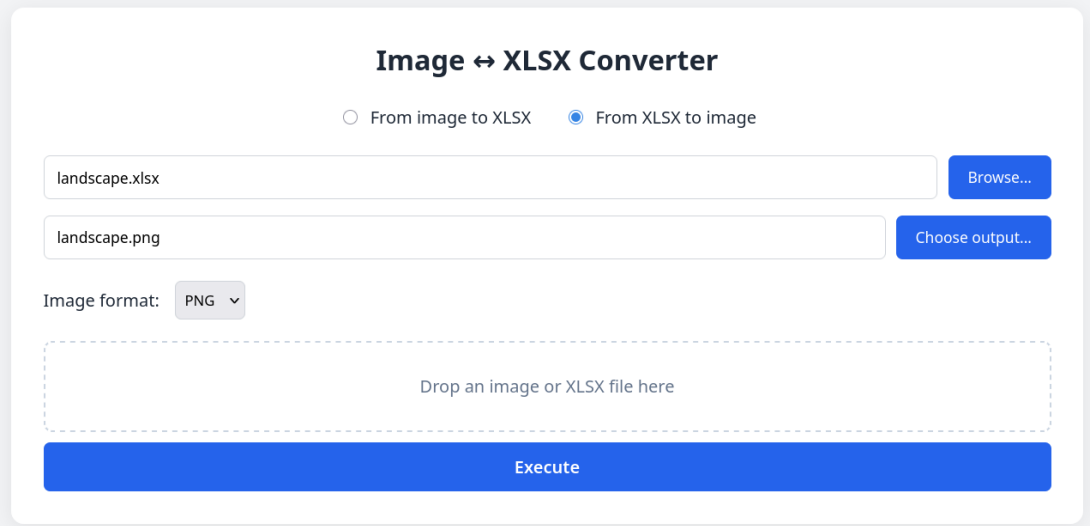
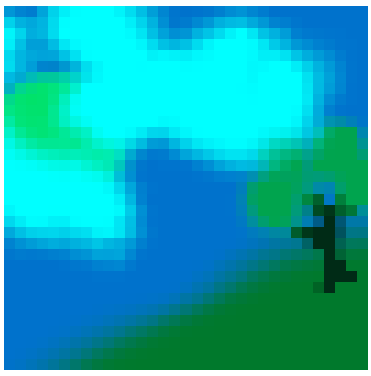
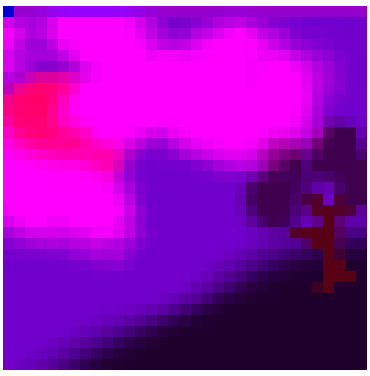


Figure 2: Landscape images with zero’ed channels

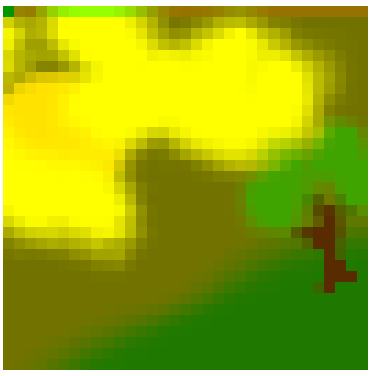
(a) Landscape image with zero’ed red channel



(b) Landscape image with zero’ed green channel



(c) Landscape image with zero’ed blue channel



Part C — `std::array`

Problem 1 — Trace the output

What does this program print?

Scale degrees

```cpp
#include <iostream>
#include <array>

int main() {
    std::array<int, 5> scale = {0, 2, 4, 5, 7};

    std::cout << scale[0]    << std::endl; // line A
    std::cout << scale[4]    << std::endl; // line B
    std::cout << scale.size() << std::endl; // line C

    scale[2] = scale[2] + 1;
    std::cout << scale[2]    << std::endl; // line D

    int total = 0;
    for (int i = 0; i < scale.size(); i = i + 1)
        total = total + scale[i];
    std::cout << total       << std::endl; // line E

    return 0;
}
```

Line A:

Line B:

Line C:

Line D:

Line E:

Problem 2 — Spot the bug

This function should return the largest value in an array of five brightness values. It contains one bug. What is it? Try and propose a fix.

Buggy maximum

```cpp
#include <array>
```

```
2
3  int max_brightness(std::array<int, 5> values) {
4      int max = 0;
5      for (int i = 1; i <= values.size(); i = i + 1) {
6          if (values[i] > max)
7              max = values[i];
8      }
9      return max;
10 }
```

Bug description:

Use: `https://godbolt.org/z/E9T8cGzY7`

Link to your fix:

Hint: What happens on the last iteration when `i == 5`?

Problem 3 — Write the code

Write a function:

```
int sum(const std::array<int, 6>& values);
```

that returns the sum of all elements in `values`. Then write a `main` that declares:

```
std::array<int, 6> volumes = {70, 85, 90, 60, 75, 80};
```

prints the sum, then prints the average (integer division).

Expected output:

```
460
76
```

Part D — `std::vector`

Problem 1 — Trace the output

What does this program print?

Building a track list

```cpp
#include <iostream>
#include <vector>

int main() {
    std::vector<int> durations;

    durations.push_back(210);
    durations.push_back(185);
    durations.push_back(240);

    std::cout << durations.size() << std::endl; // line A
    std::cout << durations[1]   << std::endl; // line B

    durations[0] = durations[0] + 15;
    std::cout << durations[0]   << std::endl; // line C

    durations.push_back(durations[2] - 10);
    std::cout << durations.size() << std::endl; // line D
    std::cout << durations[3]   << std::endl; // line E

    return 0;
}
```

Line A:

Line B:

Line C:

Line D:

Line E:

Problem 2 — What does this return?

Vector accumulation

```cpp
#include <vector>

int count_above(const std::vector<int>& values, int threshold) {
    int count = 0;
    for (int i = 0; i < values.size(); i = i + 1) {
        if (values[i] > threshold)
            count = count + 1;
    }
    return count;
}

int main() {
    std::vector<int> pixels = {30, 180, 90, 220, 15, 200, 110};
    return count_above(pixels, 100);
}
```

Return value:

Modify the code to list the elements that are above the threshold and write them here:

Problem 3 — Write the code

Write a complete program that:

1. Reads an arbitrary number of integer scores from `argv` (one per argument).
2. Stores them in a `std::vector<int>`.
3. Prints the minimum and maximum values found.

If no arguments are given, print `"no scores"` and return `1`.

Expected output for `./scores 40 95 72 18 88`:

```
min: 18
max: 95
```

Part E — 2D `std::array`

Problem 1 — Trace the output

What does this program print?

2x4 image

```cpp
#include <iostream>
#include <array>

int main() {
    std::array<std::array<int, 4>, 2> image = {{
        {10, 50, 90, 130},
        {20, 60, 100, 140}
    }};

    std::cout << image[0][0] << std::endl; // line A
    std::cout << image[1][3] << std::endl; // line B
    std::cout << image[0][2] + image[1][1] << std::endl; // line C

    int total = 0;
    for (int row = 0; row < 2; row = row + 1)
        for (int col = 0; col < 4; col = col + 1)
            total = total + image[row][col];
    std::cout << total << std::endl;      // line D

    return 0;
}
```

Line A:

Line B:

Line C:

Line D:

Problem 2 — Write the code

Write a function:

```cpp
void print_grid(const std::array<std::array<int, 4>, 3>& grid);
```

that prints each row on its own line, values separated by spaces. Then write a `main` that declares the following grid and calls `print_grid`:

```
std::array<std::array<int, 4>, 3> grid = {{
    {1,  2,  3,  4},
    {5,  6,  7,  8},
    {9, 10, 11, 12}
}};
```

Expected output:

```
1 2 3 4
5 6 7 8
9 10 11 12
```

Part F — 2D `std::vector`

Problem 1 — Trace the output

What does this program print?

Building a 2D vector

```
#include <iostream>
#include <vector>

int main() {
    std::vector<std::vector<int>> frame;

    frame.push_back({255, 128, 0});
    frame.push_back({64, 32, 16});
    frame.push_back({200, 200, 200});

    std::cout << frame.size()    << std::endl; // line A
    std::cout << frame[0].size() << std::endl; // line B
    std::cout << frame[1][2]     << std::endl; // line C

    frame[2][0] = frame[2][0] / 2;
    std::cout << frame[2][0]     << std::endl; // line D

    int total = 0;
    for (int r = 0; r < frame.size(); r = r + 1)
        for (int c = 0; c < frame[r].size(); c = c + 1)
            total = total + frame[r][c];
    std::cout << total           << std::endl; // line E

    return 0;
}
```

Line A:

Line B:

Line C:

Line D:

Line E:

Problem 2 — Spot the bug

This function should print every element of a 2D vector, row by row. It contains bugs.

Buggy 2D print

```cpp
#include <iostream>
#include <vector>

void print_image(const std::vector<std::vector<int>>& img) {
    for (int row = 0; row < img.size(); row = row + 1) {
        for (int col = 0; col <= img[row].size(); col = col + 1) {
            std::cout << img[col][row] << " ";
        }
        std::cout << std::endl;
    }
}
```

Bugs:

Use : `https://godbolt.org/z/Kehxn6qe8`

Url of fix:

Problem 3 — Write the code

Write a function:

```cpp
std::vector<std::vector<int>> make_gradient(int rows, int cols);
```

that returns a 2D vector of size `rows` × `cols` where each element equals `row * cols + col` (the flat index of that element).

Then write a `main` that calls `make_gradient(3, 4)` and prints the result row by row, values separated by spaces.

Expected output:

```
0 1 2 3
4 5 6 7
8 9 10 11
```

Part G — Mixing everything

Problem 1 — Nested loop with condition

Write a function:

```
int count_bright(const std::vector<std::vector<int>>& image, int threshold);
```

that counts how many pixels in the 2D image are strictly above `threshold`.

Then write a `main` that builds the following image, calls `count_bright` with a threshold of `100`, and prints the result.

```
std::vector<std::vector<int>> image = {
    {50, 200, 30},
    {180, 90, 240},
    {10, 130, 70}
};
```

Expected output:

Problem 2 — Command-line image builder (harder)

Write a complete program invoked as:

```
./build <rows> <cols> <fill>
```

It must:

1. Validate that exactly 3 arguments are provided; otherwise print a usage message and return `1`.
2. Read `rows`, `cols`, and `fill` using `atoi`.
3. Build a `std::vector<std::vector<int>>` of the given dimensions where every element equals `fill`.
4. Print the grid row by row, values separated by spaces.
5. Print the total sum of all elements on a final line.

Expected output for `./build 2 3 50`:

```
50 50 50
50 50 50
300
```