Introduction to Programming

Lecture 3 — Collections and sequences

Nicolas Gry

Université de Saint-Étienne | Year 2026/2027

digicrea digital creativity
Université Jean Monnet Saint-Étienne

-1 Recap time

About functions

- They enable reusing pieces of code

About functions

- They enable reusing pieces of code
- They have a signature that signals:

About functions

- They enable reusing pieces of code
- They have a signature that signals:
  - Their return type

About functions

- They enable reusing pieces of code
- They have a signature that signals:
  - Their return type
  - Their name

About functions

- They enable reusing pieces of code
- They have a signature that signals:
  - Their return type
  - Their name
  - Their arguments

About functions

- They enable reusing pieces of code
- They have a signature that signals:
  - Their return type
  - Their name
  - Their arguments
- They can be *overloaded*

About functions

- They enable reusing pieces of code
- They have a signature that signals:
  - Their return type
  - Their name
  - Their arguments
- They can be *overloaded*
  - By their number of arguments

About functions

- They enable reusing pieces of code
- They have a signature that signals:
  - Their return type
  - Their name
  - Their arguments
- They can be *overloaded*
  - By their number of arguments
  - By the type of their arguments

About functions

- They enable reusing pieces of code
- They have a signature that signals:
  - Their return type
  - Their name
  - Their arguments
- They can be *overloaded*
  - By their number of arguments
  - By the type of their arguments
  - By the constness of reference arguments

About variables scope

- Global variables are declared outside of any function

About variables scope

- Global variables are declared outside of any function
- They accessible anywhere in the code

About variables scope

- Global variables are declared outside of any function
- They accessible anywhere in the code

- Local variables are declared in functions and/or in blocks

About variables scope

- Global variables are declared outside of any function
- They accessible anywhere in the code

- Local variables are declared in functions and/or in blocks
- Their access is limited to the function/block they’re in, from the moment they’re instanciated

About variables scope

- Global variables are declared outside of any function
- They accessible anywhere in the code

- Local variables are declared in functions and/or in blocks
- Their access is limited to the function/block they’re in, from the moment they’re instanciated
- They can *shadow* global variables

About variables scope

- Global variables are declared outside of any function
- They accessible anywhere in the code

- Local variables are declared in functions and/or in blocks
- Their access is limited to the function/block they're in, from the moment they're instanciated
- They can *shadow* global variables
- In that case, the global variable is accessible by prepending `::`

About the `main` function

- `main` can have one of two signatures

About the `main` function

- `main` can have one of two signatures

```
int main();
int main(int argc, char * argv[]);
```

- `argc` stands for argument count

About the `main` function

- `main` can have one of two signatures

```
int main();
int main(int argc, char * argv[]);
```

- `argc` stands for argument count
- `argv` stands for argument value

About the `main` function

- `main` can have one of two signatures

```
int main();
int main(int argc, char * argv[]);
```

- `argc` stands for argument count
- `argv` stands for argument value
- The first argument of a program is always the invocation of that program

About the `main` function

- `main` can have one of two signatures

```
int main();
int main(int argc, char * argv[]);
```

- `argc` stands for argument count
- `argv` stands for argument value
- The first argument of a program is always the invocation of that program
- Arguments can be accessed in `argv` by using `argv[argument number]`

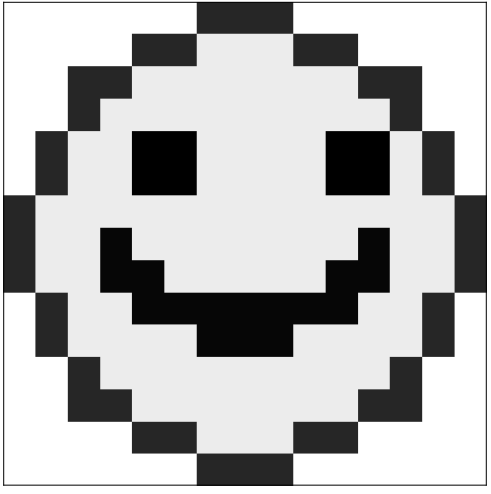
00 Assignment review

01 Introduction

What do you see? 1/2

230	230	230	230	230	230	40	40	40	230	230	230	230	230	230
230	230	230	230	40	40	195	195	195	40	40	230	230	230	230
230	230	40	40	195	195	195	195	195	195	195	40	40	230	230
230	230	40	195	195	195	195	195	195	195	195	195	40	230	230
230	40	195	195	10	10	195	195	195	195	10	10	195	40	230
230	40	195	195	10	10	195	195	195	195	10	10	195	40	230
40	195	195	195	195	195	195	195	195	195	195	195	195	195	40
40	195	195	15	195	195	195	195	195	195	195	15	195	195	40
40	195	195	15	15	195	195	195	195	195	15	15	195	195	40
230	40	195	195	15	15	15	15	15	15	15	195	195	40	230
230	40	195	195	195	195	15	15	15	195	195	195	195	40	230
230	230	40	195	195	195	195	195	195	195	195	195	40	230	230
230	230	40	40	195	195	195	195	195	195	195	40	40	230	230
230	230	230	230	40	40	195	195	195	40	40	230	230	230	230
230	230	230	230	230	230	40	40	40	230	230	230	230	230	230

What do you see? 2/2



Arrays and matrices everywhere

Arrays and matrices everywhere

Images are stored as matrices

Arrays and matrices everywhere

Images are stored as matrices

Sound is stored as arrays

Arrays and matrices everywhere

Images are stored as matrices

Sound is stored as arrays

Even text files can be viewed as matrices

02 Matrices

Excel

Has everyone already used / opened excel once in their life?

O-Spreadsheet

Let’s explore matrices with O-Spreadsheet

Color theory

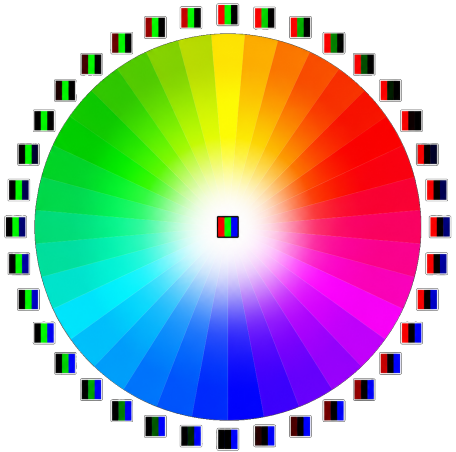


Figure: Color theory illustrated with a color wheel

Matrices shape and indexing

Matrices have a shape usually noted *(rows, columns)*.

Matrices shape and indexing

Matrices have a shape usually noted *(rows, columns)*.
Elements of a matrix are usually **indexed** as *(row number, column number) — it's reversed in spreadsheets*.

Matrices shape and indexing

Matrices have a shape usually noted *(rows, columns)*.

Elements of a matrix are usually **indexed** as *(row number, column number) — it's reversed in spreadsheets*.

You can stack matrices to get a 3rd dimension.

Matrices shape and indexing

Matrices have a shape usually noted *(rows, columns)*.

Elements of a matrix are usually **indexed** as *(row number, column number) — it's reversed in spreadsheets*.

You can stack matrices to get a 3rd dimension. The shape is then *(channel, row, column)*.

Matrices shape and indexing

Matrices have a shape usually noted *(rows, columns)*.

Elements of a matrix are usually **indexed** as *(row number, column number) — it's reversed in spreadsheets*.

You can stack matrices to get a 3rd dimension. The shape is then *(channel, row, column)*.

Element indexing is then *(channel number, row number, column number)*.

Matrices shape and indexing

Matrices have a shape usually noted *(rows, columns)*.

Elements of a matrix are usually **indexed** as *(row number, column number) — it's reversed in spreadsheets*.

You can stack matrices to get a 3rd dimension. The shape is then *(channel, row, column)*.

Element indexing is then *(channel number, row number, column number)*.

For a rgb-image, this translates to *(color channel(r,g,b), pixel x coordinate, pixel y coordinate)*.

Matrices shape and indexing: examples

Matrices shape and indexing: examples

4	23	18	23	9
2	30	20	4	1
13	10	22	23	0

Shape? Location of 18? Location of 0?

Matrices shape and indexing: examples

4	23	18	23	9
2	30	20	4	1
13	10	22	23	0

Shape? Location of 18? Location of 0?

Shape: (3, 5)

Matrices shape and indexing: examples

4	23	18	23	9
2	30	20	4	1
13	10	22	23	0

Shape? Location of 18? Location of 0?

Shape: (3, 5)

18 is at (1, 3)

Matrices shape and indexing: examples

4	23	18	23	9
2	30	20	4	1
13	10	22	23	0

Shape? Location of 18? Location of 0?

Shape: (3, 5)

18 is at (1, 3)

0 is at (3, 5)

Matrices shape and indexing: examples

4	23	18	23	9
2	30	20	4	1
13	10	22	23	0

Shape? Location of 18? Location of 0?

Shape: (3, 5)

18 is at (1, 3)

0 is at (3, 5)

thrift	adult	fiscally
aerugo	diotrephes	knit
repay	pentecostal	clears
scrofulous	cumuliform	regional

Shape? Location of “knit”? Location of “cumuliform”?

Matrices shape and indexing: examples

4	23	18	23	9
2	30	20	4	1
13	10	22	23	0

Shape? Location of 18? Location of 0?

Shape: (3, 5)

18 is at (1, 3)

0 is at (3, 5)

thrift	adult	fiscally
aerugo	diotrephes	knit
repay	pentecostal	clears
scrofulous	cumuliform	regional

Shape? Location of “knit”? Location of “cumuliform”?

Shape: (4, 3)

Matrices shape and indexing: examples

4	23	18	23	9
2	30	20	4	1
13	10	22	23	0

Shape? Location of 18? Location of 0?

Shape: (3, 5)

18 is at (1, 3)

0 is at (3, 5)

thrift	adult	fiscally
aerugo	diotrephes	knit
repay	pentecostal	clears
scrofulous	cumuliform	regional

Shape? Location of “knit”? Location of “cumuliform”?

Shape: (4, 3)

“knit” is at (3, 2)

Matrices shape and indexing: examples

4	23	18	23	9
2	30	20	4	1
13	10	22	23	0

Shape? Location of 18? Location of 0?

Shape: (3, 5)

18 is at (1, 3)

0 is at (3, 5)

thrift	adult	fiscally
aerugo	diotrephes	knit
repay	pentecostal	clears
scrofulous	cumuliform	regional

Shape? Location of “knit”? Location of “cumuliform”?

Shape: (4, 3)

“knit” is at (3, 2)

“cumuliform” is at (4, 1)

Practise time!

03 Arrays in C++

Arrays

An **array** is simply a one-dimensional matrix, that is, it's a matrix of shape *(array's length, 1)*.

0-based indexing

In C++ (and in most programming languages), the **index of the first element** in an array is **0**.

0-based indexing

In C++ (and in most programming languages), the **index of the first element** in an array is **0**.

Think of it as an offset from the start of the array: the first element is zero spot away from the beginning of the array, the second is one spot away, etc.

0-based indexing

In C++ (and in most programming languages), the **index of the first element** in an array is **0**.

Think of it as an offset from the start of the array: the first element is zero spot away from the beginning of the array, the second is one spot away, etc.

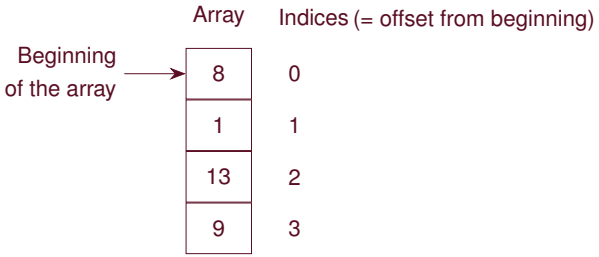


Figure: Example of an array and its indices

Array indexing examples

Array indexing examples

Size? Location of 7? Location of 0?

Array indexing examples

Size? Location of 7? Location of 0?

Size: 7

Array indexing examples

Size? Location of 7? Location of 0?

Size: 7

7 is at index 3

Array indexing examples

Size? Location of 7? Location of 0?

Size: 7

7 is at index 3

0 is at index 5

Array indexing examples

Size? Location of 7? Location of 0?

Size: 7

7 is at index 3

0 is at index 5

Array indexing examples

Size? Location of 7? Location of 0?

Size: 7

7 is at index 3

0 is at index 5

Size? Location of “reproductible”? Location of “harmonicon”?

Array indexing examples

Size? Location of 7? Location of 0?
Size: 7
7 is at index 3
0 is at index 5

Size? Location of “reproductible”? Location of “harmonicon”?
Size: 4

Array indexing examples

Size? Location of 7? Location of 0?
Size: 7
7 is at index 3
0 is at index 5

Size? Location of “reproductible”? Location of “harmonicon”?
Size: 4
“reproductible” is at index 0

Array indexing examples

Size? Location of 7? Location of 0?
Size: 7
7 is at index 3
0 is at index 5

Size? Location of “reproductible”? Location of “harmonicon”?
Size: 4
“reproductible” is at index 0
“harmonicon” is at index 3

How about some code?

Your favorite online editor

Arrays generalities

- Use `#include <array>` to have access to `std::array`.

Arrays generalities

- Use `#include <array>` to have access to `std::array`.
- Arrays have a **fixed size**: they're called *static* arrays.

Arrays generalities

- Use `#include <array>` to have access to `std::array`.
- Arrays have a **fixed size**: they're called *static* arrays.
- Arrays can contain any type of data, but exactly one type per array:

Arrays generalities

- Use `#include <array>` to have access to `std::array`.
- Arrays have a **fixed size**: they're called *static* arrays.
- Arrays can contain any type of data, but exactly one type per array:

```
std::array<std::string, 3> a1 = {"Hello,", " world", "!"}; //
    ok, array of std::string
std::array<int, 2> a2 = {"Hello,", 14}; // nok, array of int
   contains a std::string
```

Arrays generalities

- Use `#include <array>` to have access to `std::array`.
- Arrays have a **fixed size**: they're called *static* arrays.
- Arrays can contain any type of data, but exactly one type per array:

```
std::array<std::string, 3> a1 = {"Hello,", " world", "!"}; //
    ok, array of std::string
std::array<int, 2> a2 = {"Hello,", 14}; // nok, array of int
   contains a std::string
```

- The element at position i can be accessed with the operator `[]`: `arr[i]`

Practise Time!

04 Matrices in C++

What about matrices?

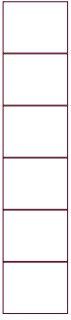
What about matrices?

Matrices can be modelled by an *array of arrays*:

`std::array<std::array<type, column_count>, line_count>`.

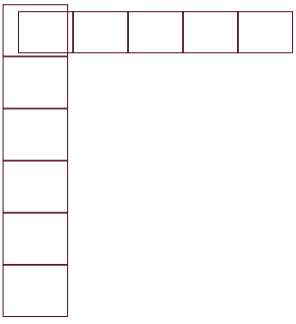
Understanding array of arrays

- Take a classic array



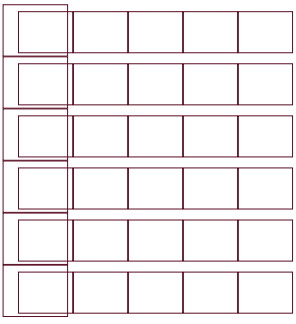
Understanding array of arrays

- Take a classic array
- If you look at its first element, it contains an array



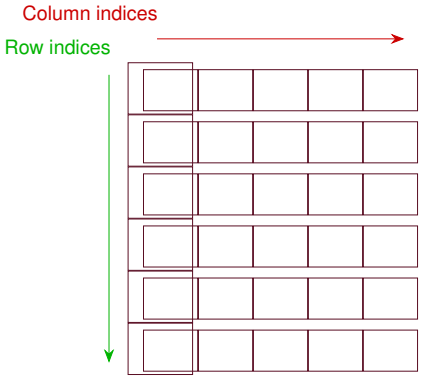
Understanding array of arrays

- Take a classic array
- If you look at its first element, it contains an array
- And so on for each element



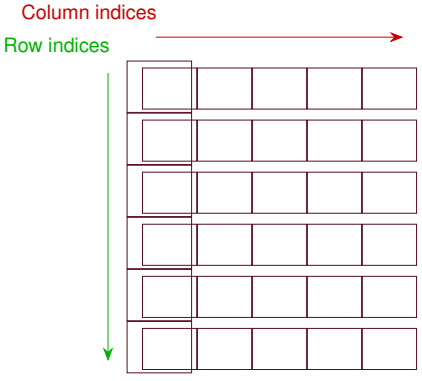
Understanding array of arrays

- Take a classic array
- If you look at its first element, it contains an array
- And so on for each element
- The end result is a matrix: you can index elements with a row index and a column index



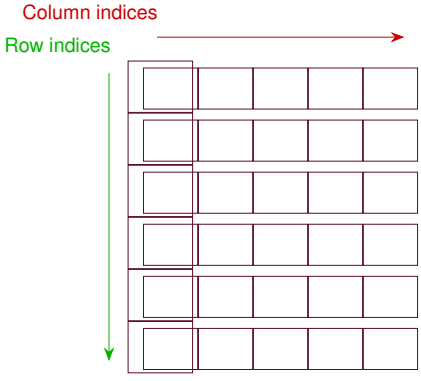
Indexing array of arrays

```
// declaration ?
```



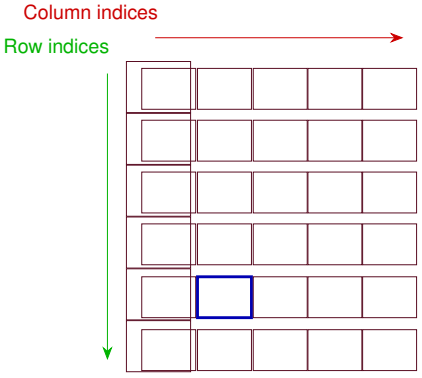
Indexing array of arrays

```
// declaration
std::array<
  std::array<int, 5>, 6>
      the_array {};
```



Indexing array of arrays

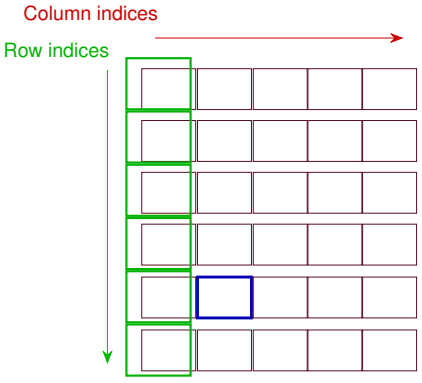
```
// declaration
std::array<
  std::array<int, 5>, 6>
      the_array {};

// indexing ?
```



Indexing array of arrays

```
// indexing ?
```

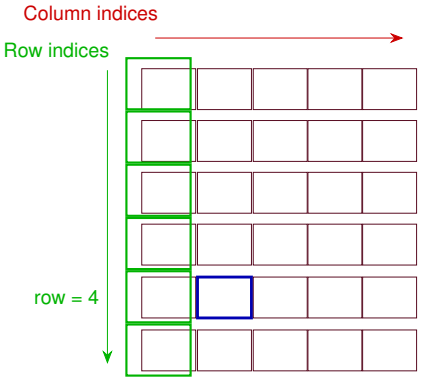
Find the row index



Indexing array of arrays

```
// indexing ?
```

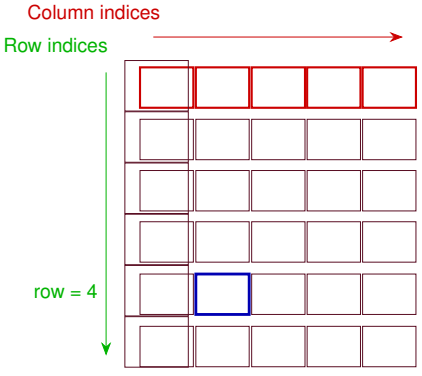
Find the row index



Indexing array of arrays

```
// indexing ?
```

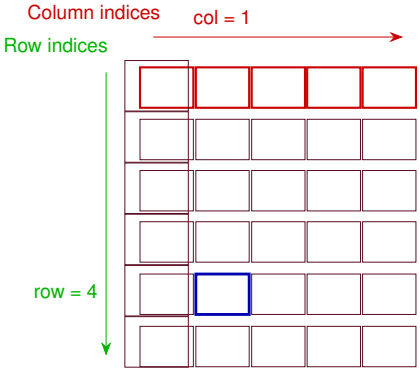
Find the column index



Indexing array of arrays

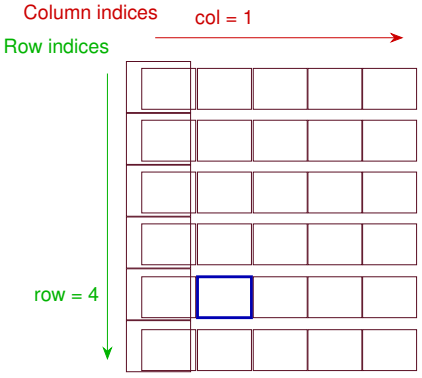
```
// indexing ?
```

Find the column index



Indexing array of arrays

```
// indexing
the_array[4][1] = ...;
```



Traversing an array of arrays

We use **nested loops**: look here.

Practise Time!

05 Vectors

Variable-length sequences

What if our sequence size is unknown and / or varies through time?
What if we can’t know in advance the amount of values we’re going to need to store?

`std::vector` to the rescue

Have a look at this code.

Generalities on `std::vector`

- Their size can vary throughout the program
- You can had an element to `std::vector vec` with its *member function* `vec.push_back(the_elem_to_add);`
- You can remove the last element of `std::vector vec` with its *member function* `vec.pop_back();`
- You can remove all elements of `std::vector vec` with its *member function* `vec.clear();`

Matrices with vectors?

Because `std::vector` has a variable size, each element of a `std::vector<std::vector<...>>`, that is, each `std::vector<...>` in that `std::vector<std::vector<...>>` may have a different length than the others.

Matrices with vectors?

Because `std::vector` has a variable size, each element of a `std::vector<std::vector<...>>`, that is, each `std::vector<...>` in that `std::vector<std::vector<...>>` may have a different length than the others.

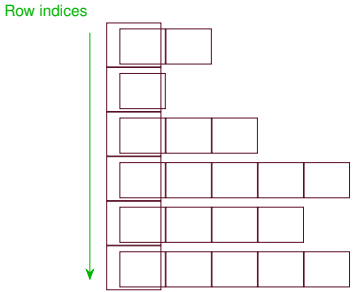


Figure: Example of a vector of vector

Nested loops for nested vectors

The inner loop range has to be recomputed at each line of the matrix by taking the size of that line.

Nested loops for nested vectors

The inner loop range has to be recomputed at each line of the matrix by taking the size of that line.

```
std::vector<std::vector<int>> vec {};
for(auto line = 0; line < vec.size(); ++line){
    for(auto col = 0; col < vec[line].size(); ++col){
        // ... do something ...
    }
}
```

Practise Time!

06 Little preparation for next time

For those with personal laptops

Try to install the following softwares:

- Visual Studio Code
- cmake
- clang

(Install instructions in the following slides.)

MacOS

Open a terminal and run the following commands:

```
xcode-select --install
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent
    .com/Homebrew/install/HEAD/install.sh)"
brew install --cask visual-studio-code
brew install cmake
```

Windows

- Install VSCode by downloading the installer from here:
  `https://code.visualstudio.com/docs?dv=win`
- Download and run *Visual Studio Community Edition*:
  `https://visualstudio.microsoft.com/thank-you-downloading-visual-studio/?sku=Community&channel=Stable&version=VS18&source=VSLandingPage&passive=false&cid=2500`
  - Follow these installation steps:

Linux

If you’re on linux, I trust you know what you’re doing. Install vscode, cmake and clang/clang++.

If you did not succeed

Since everything is up and running on the uni's computers, if you were unable to install things on your personal laptop, you can practise the next sessions on the uni's computers.

See you next time!