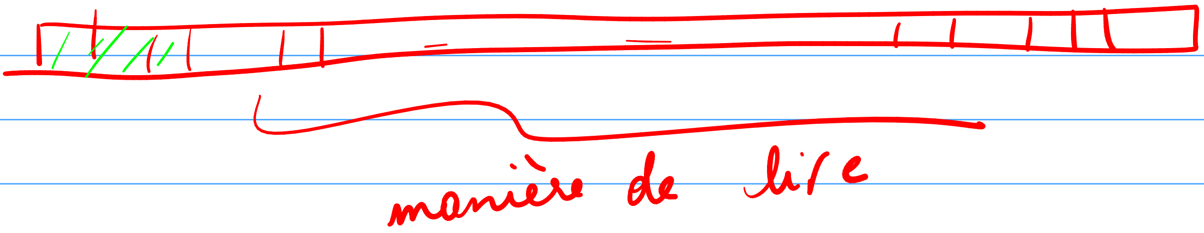
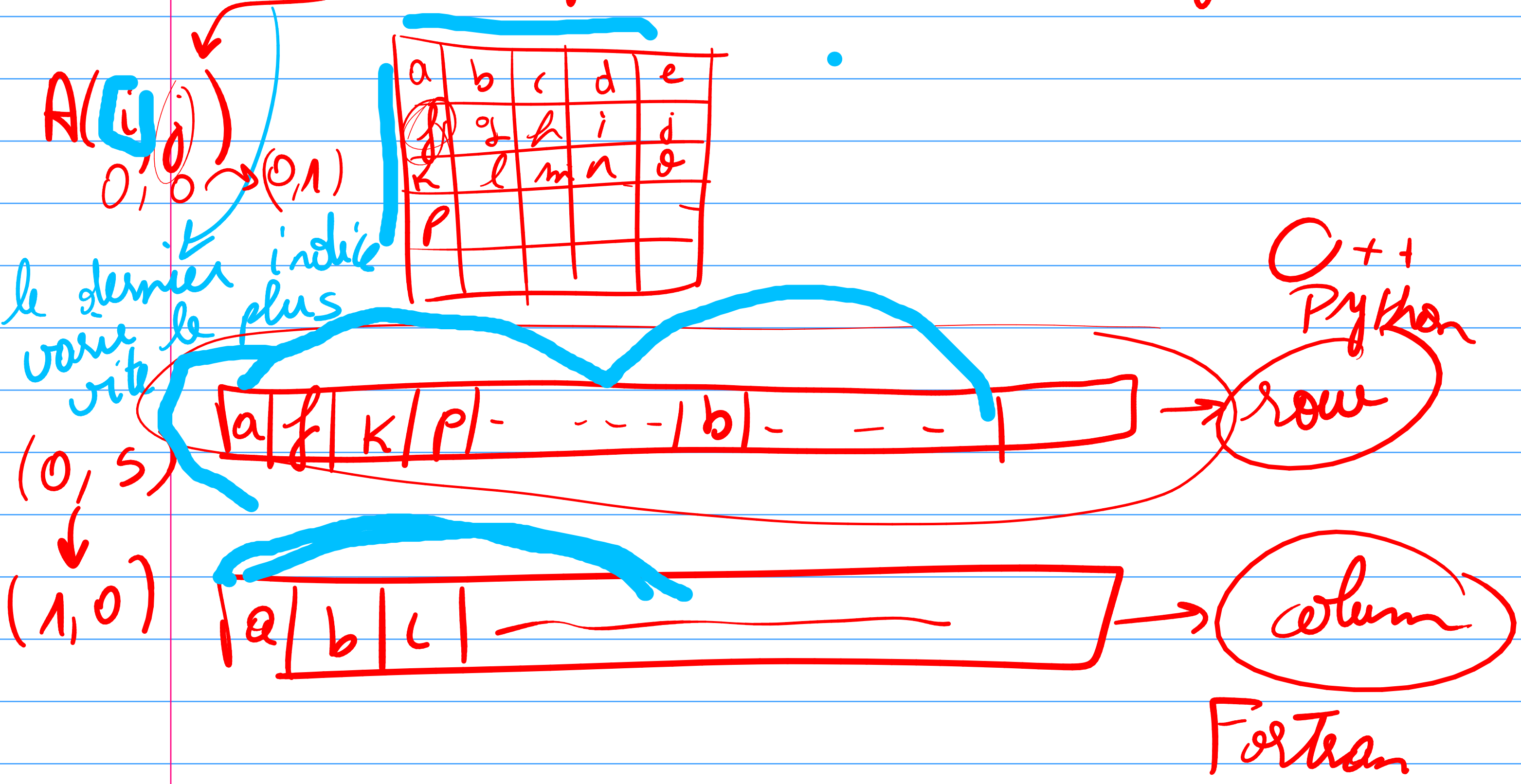
2D 3D tableau



→ contiguous memory.
consecutife

→ row-major column-major

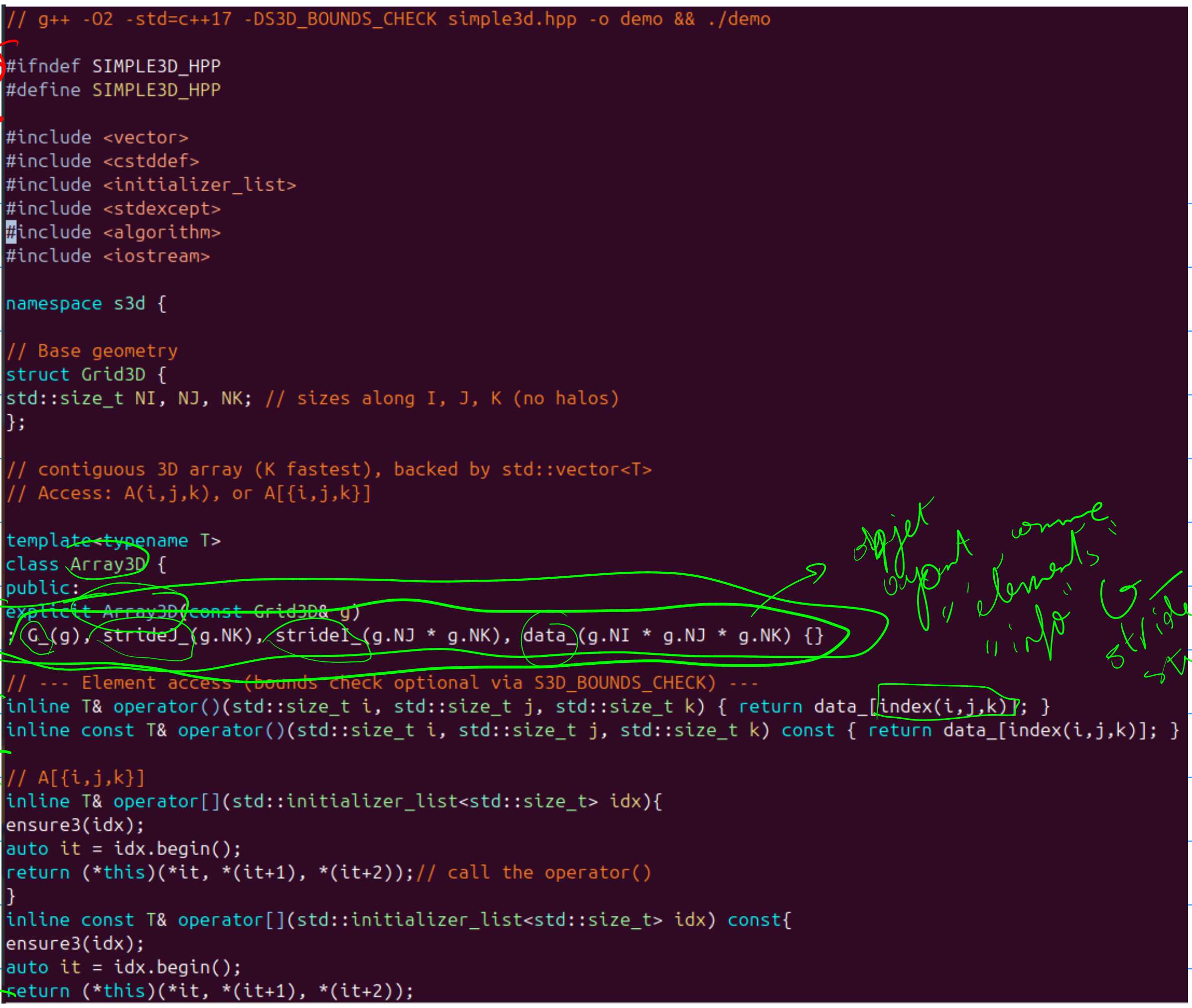


a	b	c	d	e
f	g	h	i	j
k	l	m	n	o
p				

A(i,j)
0,0 → (0,1)
le dernier indice varie le plus vite
(0,5)
(1,0)
a f k p - - - - b - - - -
a b c
C++
Python
row
colum
Fortran

Surcharge de fonctions:

On définit une $\hat{m}$ fct, plusieurs fois, pour des types différents afin de pouvoir l'utiliser avec tout ces types



```cpp
// g++ -O2 -std=c++17 -DS3D_BOUNDS_CHECK simple3d.hpp -o demo && ./demo

#ifndef SIMPLE3D_HPP
#define SIMPLE3D_HPP

#include <vector>
#include <cstddef>
#include <initializer_list>
#include <stdexcept>
#include <algorithm>
#include <iostream>

namespace s3d {

// Base geometry
struct Grid3D {
std::size_t NI, NJ, NK; // sizes along I, J, K (no halos)
};

// contiguous 3D array (K fastest), backed by std::vector<T>
// Access: A(i,j,k), or A[{i,j,k}]

template<typename T>
class Array3D {
public:
explicit Array3D(const Grid3D& g)
: G_(g), strideJ_(g.NK), strideI_(g.NJ * g.NK), data_(g.NI * g.NJ * g.NK) {}

// --- Element access (bounds check optional via S3D_BOUNDS_CHECK) ---
inline T& operator()(std::size_t i, std::size_t j, std::size_t k) { return data_[index(i,j,k)]; }
inline const T& operator()(std::size_t i, std::size_t j, std::size_t k) const { return data_[index(i,j,k)]; }

// A[{i,j,k}]
inline T& operator[](std::initializer_list<std::size_t> idx){
ensure3(idx);
auto it = idx.begin();
return (*this)(*it, *(it+1), *(it+2));// call the operator()
}
inline const T& operator[](std::initializer_list<std::size_t> idx) const{
ensure3(idx);
auto it = idx.begin();
return (*this)(*it, *(it+1), *(it+2));
```

#endif (à la fin)

objet ayant comme "elements" "info" G stride I stride J data.

structure (classes) → groupé ensemble des informations qui se ressemblet

structure → groupe simple

classes → groupes

std :: Initializer_list <T> :

une fct qui donne accés à un tableau de valeurs de type (T)

.begin () → le début du tableau

.end () → la fin du tableau

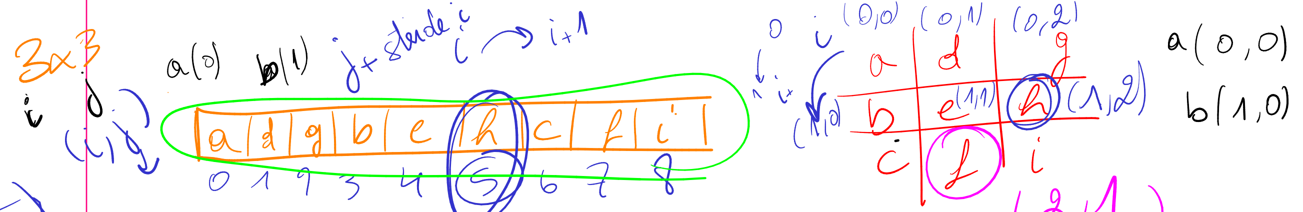
.size () → combien d'elt

auto : type que C++ choisi automatiquement

int a = 4;

auto b = a;

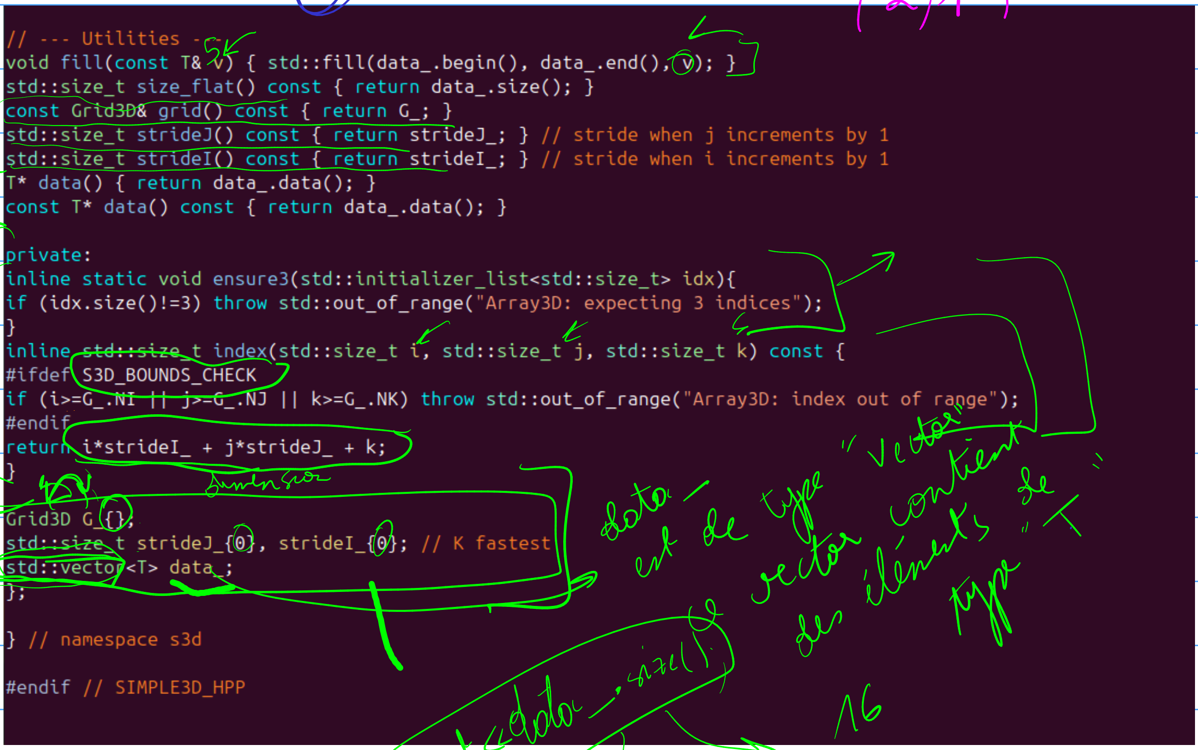
type de b : int



a(0,0)
b(1,0)

(1,2)
stride j=1
stride i=3

g++



```
// --- Utilities -
void fill(const T& v) { std::fill(data_.begin(), data_.end(), v); }
std::size_t size_flat() const { return data_.size(); }
const Grid3D& grid() const { return G_; }
std::size_t strideJ() const { return strideJ_; } // stride when j increments by 1
std::size_t strideI() const { return strideI_; } // stride when i increments by 1
T* data() { return data_.data(); }
const T* data() const { return data_.data(); }

private:
inline static void ensure3(std::initializer_list<std::size_t> idx){
if (idx.size()!=3) throw std::out_of_range("Array3D: expecting 3 indices");
}
inline std::size_t index(std::size_t i, std::size_t j, std::size_t k) const {
#ifdef S3D_BOUNDS_CHECK
if (i>=G_.NI || j>=G_.NJ || k>=G_.NK) throw std::out_of_range("Array3D: index out of range");
#endif
return i*strideI_ + j*strideJ_ + k;
}

Grid3D G_{};
std::size_t strideJ_{0}, strideI_{0}; // K fastest
std::vector<T> data_;
};

} // namespace s3d

#endif // SIMPLE3D_HPP
```

dimension

data est de type "vector" contient des éléments de type "T"

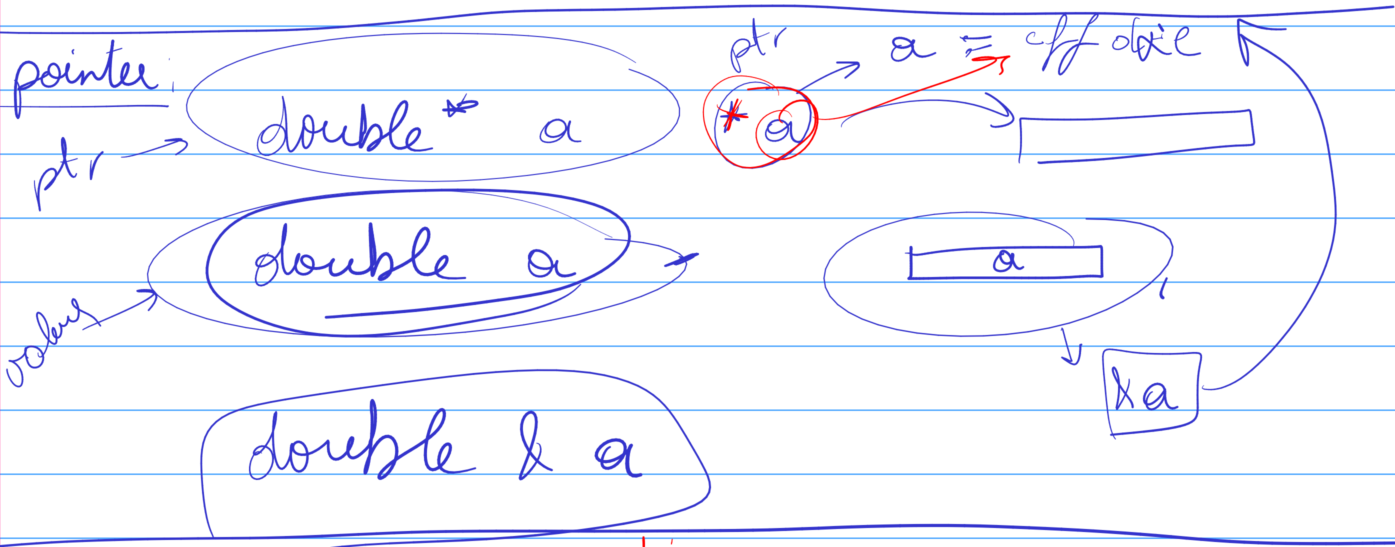
cout << data_.size()

16

.begin
.end

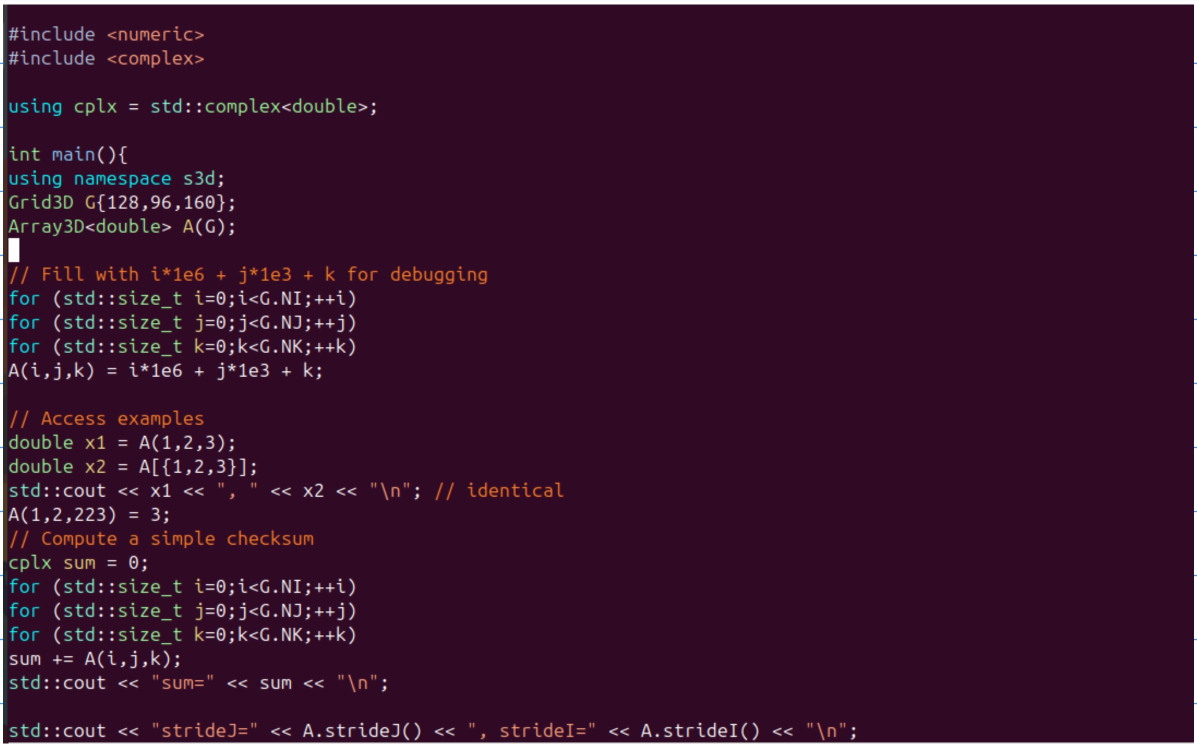
2D (i,j) j

3D (i,j,k)



class → obj
↘ méthodes (fct) → ça agit sur un obj

this → l'adress de l'objet sur lequel cette fct agit



```
#include <numeric>
#include <complex>

using cplx = std::complex<double>;

int main(){
using namespace s3d;
Grid3D G{128,96,160};
Array3D<double> A(G);

// Fill with i*1e6 + j*1e3 + k for debugging
for (std::size_t i=0;i<G.NI;++i)
for (std::size_t j=0;j<G.NJ;++j)
for (std::size_t k=0;k<G.NK;++k)
A(i,j,k) = i*1e6 + j*1e3 + k;

// Access examples
double x1 = A(1,2,3);
double x2 = A[{1,2,3}];
std::cout << x1 << ", " << x2 << "\n"; // identical
A(1,2,223) = 3;
// Compute a simple checksum
cplx sum = 0;
for (std::size_t i=0;i<G.NI;++i)
for (std::size_t j=0;j<G.NJ;++j)
for (std::size_t k=0;k<G.NK;++k)
sum += A(i,j,k);
std::cout << "sum=" << sum << "\n";

std::cout << "strideJ=" << A.strideJ() << ", strideI=" << A.strideI() << "\n";
```