

# Arrays in Classes

Victor Eijkhout, Susan Lindsey

Fall 2025

last formatted: October 2, 2025

## Vectors in classes

# 1. Can you make a class around a vector?

You may want a class of objects that contain a vector. For instance, you may want to name your vectors.

```
1 class named_field {  
2 private:  
3     string name;  
4     vector<double> values;
```

The problem here is when and how that vector is going to be created.

## 2. Create the contained vector

Use initializers for creating the contained vector:

```
1 class named_field {
2 private:
3     string name;
4     vector<double> values;
5 public:
6     named_field( string name,int n )
7         : name(name),
8           values(vector<double>(n)) {
9     };
10};
```

Even shorter:

```
1 named_field( string name,int n )
2     : name(name),values(n) {
3     };
```

## Multi-dimensional arrays

### 3. Multi-dimensional vectors

Multi-dimensional is harder with vectors:

```
1 vector<float> row(20);  
2 vector<vector<float>> rows(10,row);
```

Create a row vector, then store 10 copies of that:  
vector of vectors.

## 4. Naive matrix class

### Simplest implementation

```
1 // array/matrixclass.cpp
2 class matrix {
3 private:
4     vector<vector<double>>
        matrix_data;
5     int m,n;
6 public:
7     matrix(int m,int n)
8         : m(m),n(n) {
9         matrix_data = vector<
10             vector<double>(n)
11             >(m);
12     };
13 };
```

### Better constructor

```
1 // array/matrixclass.cpp
2 matrix(int m,int n)
3     : m(m),n(n),
4       matrix_data(
5           vector<vector<double>(n)>(m)
6           ) {};
```

## 5. Implementation of a vector

```
std::vector<float>(10);
```

size\_t size = 10  
float \*data

(on the stack)

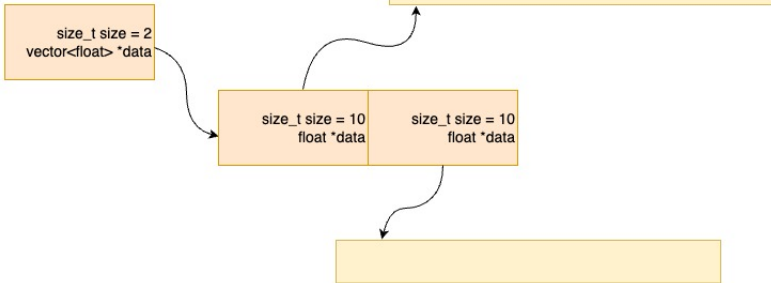
data

(on the heap)



## 6. Vector of vectors

```
std::vector<vector<float>>(2,vector<float>(10));
```



## 7. Better matrix class

```
1 // array/matrixclass.cpp
2 class matrix {
3 private:
4     vector<double> matrix_data;
5     int m,n;
6 public:
7     matrix(int m,int n)
8         : m(m),n(n),matrix_data(m*n) {};
```

Later we will write methods `set(i,j,v)` and `get(i,j)`

# Exercise 1

Write `rows()` and `cols()` methods for this class that return the number of rows and columns respectively.

## Exercise 2

Write a method `void set(double)` that sets all matrix elements to the same value.

Write a method `double totalsum()` that returns the sum of all elements.

Code:

```
1 // array/matrix.cpp
2 A.set(3.);
3 cout << "Sum of elements: "
4     << A.totalsum() << '\n';
```

Output:

```
1 Sum of elements: 30
```

## 8. Matrix class; better design

Linearized indexing:  
Class:

```
1 // array/matrixclass.cpp
2 class matrix {
3 private:
4     vector<double> matrix_data;
5     int m,n;
6 public:
7     matrix(int m,int n)
8         : m(m),n(n),matrix_data(m*n)
9         {};
```

Methods:

```
1 void setij(int i,int j,double v) {
2     matrix_data.at( i*n +j ) = v;
3 };
4 double getij(int i,int j) {
5     return matrix_data.at( i*n +j );
6 };
```

## Exercise 3

Can you set/get elements by overloading the index operator?

## Exercise 4

In the matrix class of the previous slide, why are  $m, n$  stored explicitly while that would not be needed in the scheme of section ???

## Exercise 5

Add methods such as *transpose*, *scale* to your matrix class.

Implement matrix-matrix multiplication.



## 9. Pascal's triangle

Pascal's triangle contains binomial coefficients:

Row	1:																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			
-----	----	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

where

$$p_{rc} = \binom{r}{c} = \frac{r!}{c!(r-c)!}.$$

The coefficients can be computed from the recurrence

$$p_{rc} = \begin{cases} 1 & c \equiv 1 \vee c \equiv r \\ p_{r-1,c-1} + p_{r-1,c} & \text{otherwise} \end{cases}$$

## Exercise 6

- Write a class `pascal` so that `pascal(n)` is the object containing  $n$  rows of the above coefficients. Write a method `getvalue(i,j)` that returns the  $(i,j)$  coefficient.
- Write a method `print` that prints the above display.

The object needs to have an array internally. The easiest solution is to make an array of size  $n \times n$ . Optionally you can optimize your code to use precisely enough space for the coefficients.

Write a method `print(int m)` that prints a star if the coefficient modulo  $m$  is nonzero, and a space otherwise.

```

      *
    * *
  *   *
 * * * *
*   *   *
 * *       *
  * *     * *
    *   *   *   *
      * * * * * * *
        *           *
      * *         * *
```

# Optional exercise 7

Extend the Pascal exercise:

Optimize your code to use precisely enough space for the coefficients.

Write a class *storage* that provides *get/set* methods that only read from and write to the data structure. The *pascal* class can then inherit from it, and do the coefficient calculation. Do you use public or private inheritance? Extend the *storage* class:

- If a coefficient outside the initial triangle is asked, the triangle should dynamically be extended to the row of that coefficient.
- This requires the *storage* class to extend the space for the coefficients.
- It also requires the *pascal* class to track how many rows have been filled in, and possibly compute some missing coefficients.

## Other array stuff

## 10. Array class

Arrays:

```
1 #include <array>
2 std::array<int,5> fiveints;
```

- Size known at compile time.
- Vector methods that do not affect storage
- Zero overhead.

# 11. Random walk exercise

```
// walk/walk_lib_vec.cpp
class Mosquito {
private:
    vector<float> pos;
public:
    Mosquito( int d )
        : pos( vector<float>(d,0.f) ) { };

// walk/walk_lib_vec.cpp
void step() {
    int d = pos.size();
    auto incr = random_step(d);
    for (int id=0; id<d; ++id)
        pos.at(id) += incr.at(id);
};
```

Finish the implementation. Do you get improvement from using the array class?



## 12. Span

Create a `span` from a `vector`, starting at its second element and ending before its last:

```
1 #include <span>
2 vector<double> v;
3 std::span<double> v_span( v.data()+1, v.size()-2 );
```

# Array creation

C-style arrays still exist,

```
// array/staticinit.cpp
{
    int numbers[] = {5,4,3,2,1};
    println("{} ", numbers[3]);
}
{
    int numbers[5]{5,4,3,2,1};
    numbers[3] = 21;
    println("{} ", numbers[3]);
}
```

but you shouldn't use them.

Prefer to use `array` class (not in this course)

or `span` (C++20; very advanced)