CS319: Scientific Computing

Week 7: Multidimensional Arrays, and Analysis of Algorithms

Dr Niall Madden

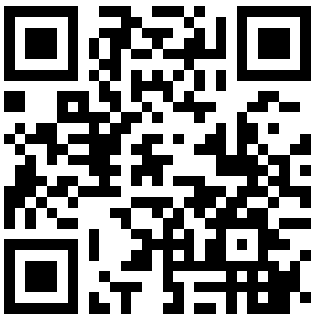
9am and **4pm**, 21 February, 2024

Slides and examples: https://www.niallmadden.ie/2324-CS319

Annotated slides from 4pm class.
However, much of that class was spent working through CS319-Week07.ipynb

Outline

Slides and examples:
https://www.niallmadden.ie/2324-CS319



Recall the idea of a one dimensional array. For example, if we wanted to store a set of **five** `int`egers, we could declare an array

```
int v[5];
```

We could then access the five elements:
`v[0]`, `v[1]`, `v[2]`, `v[3]`, and `v[4]`.

This is a **one-dimensional array**: the array has a single index. It is similar to the idea of a **vector** in Mathematics.

Often we will have table/rectangles of data, in a way that is similar to a **matrix**.

Two-dimensional arrays

In C++, an two-dimensional $M \times N$ array of (say) `doubles` can be declared as:

```
double A[M][N];
```

Then its members are

$$\begin{pmatrix} \texttt{A[0][0]} & \texttt{A[0][1]} & \texttt{A[0][2]} & \cdots & \texttt{A[0][N-1]} \\ \texttt{A[1][0]} & \texttt{A[1][1]} & \texttt{A[1][2]} & \cdots & \texttt{A[1][N-1]} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \texttt{A[M-1][0]} & \texttt{A[M-1][1]} & \texttt{A[M-1][2]} & \cdots & \texttt{A[M-1][N-1]} \end{pmatrix}$$

For example, `double A[3][4];` gives a 2D array

$$\begin{pmatrix} \texttt{A[0][0]} & \texttt{A[0][1]} & \texttt{A[0][2]} & \texttt{A[0][3]} \\ \texttt{A[1][0]} & \texttt{A[1][1]} & \texttt{A[1][2]} & \texttt{A[1][3]} \\ \texttt{A[2][0]} & \texttt{A[2][1]} & \texttt{A[2][2]} & \texttt{A[2][3]} \end{pmatrix}$$

Dynamic memory allocation in 2D is a little complicated, because a 2D array is actually just an "array of arrays".

This is because, we declare, for example,
`double A[3][4];` what really happens is:

- `A` is assigned the base address of three **pointers**: `A[0]`, `A[1]`, `A[2]`.
- Each of those is a base address for a (1D) array of 4 doubles.

This approach has advantages: because of it the language can support arrays in as many dimensions as one would like.

But it makes DMA more complicated.

To use dynamic memory allocation to reserve memory for a two-dimensional $M \times N$ matrix of doubles (for example):

- declare a "pointer to pointer to double"
- use `new` to assign memory for M pointers;
- for each of those, assign memory for N doubles.

Code:

```
double **A;
A = new double* [M];
for (int i=0; i<M; i++)
      A[i] = new double [N];
```

Finished here at 5pm!