Intermediate Scientific Computing

Week 17

Structures and Objects in C++

University of BRISTOL

Dr Stephen Clark

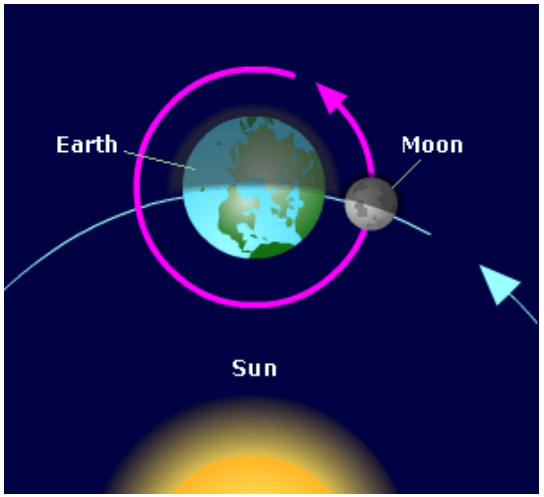
Structures and Objects in C++

The purpose of this workshop is to briefly introduce structures and objects in C++. The *learning objectives* are:

- To learn how to define and use *structures*.
- To appreciate how *object classes* are defined in C++

What are structures?

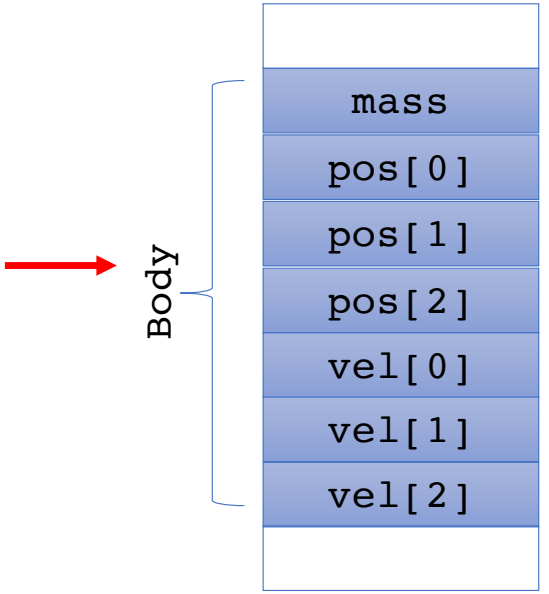
Structures are ways of bundling together variables (of different types) and access them by their names rather than indices. Here's an example:




```
1    double sunPos[3], moonPos[3], earthPos[3]; // Position vectors
2    double sunVel[3], moonVel[3], earthVel[3]; // Velocity vectors
3    double sunMass, earthMass, moonMass; // Masses
```

Instead, we can define:

```
1  struct body {
2      double mass; // Mass
3      double pos[3]; // Position vector
4      double vel[3]; // Velocity vector
5  };
```



and then declare three instances:

```
1    body sun, moon, earth; // Declare structures for all three bodies
```

Accessing and modifying members

Define outside `main()`

Declare in `main()`

Assign via `.` notation

```
1  #include <iostream>
2
3  // Define the structure outside and before the main() function
4  struct body {
5      double mass; // Mass
6      double pos[3]; // Position vector
7      double vel[3]; // Velocity vector
8  };
9
10 int main()
11 {
12     Body sun, moon, earth; // Declare structures for all three bodies
13
14     earth.mass = 6.0e24; // Mass of earth [kg]
15     // Set the sun as the origin of our coordinate system:
16     sun.pos[0] = 0.0;
17     sun.pos[1] = 0.0;
18     sun.pos[2] = 0.0;
19
20     // and so on ...
21
22     return EXIT_SUCCESS;
23 }
```

A brief outlook on Objects in C++

Structures in C are the precursor to full OOP in C++ called classes.

functions (called methods) are defined within the class to access and modify its data

like a structure but the data is now **private**

```
1  /* --- Declaration of rational --- */
2
3  class rational {
4      public:
5          // Constructors
6          rational();
7          rational(int a, int b);
8          // Responsibilities
9          void print() const;
10         rational operator+(const rational& other_rational) const;
11     private:
12         int numerator; // The numerator
13         int denominator; // The denominator
14 };
```

The class methods declared then need to be implemented …

A brief outlook on Objects in C++

The most significant method in this example overloading the “+” operator:

```
23 rational rational::operator+(const rational& other_rational) const
24 {
25     // Compute the addition of two rational numbers:
26     int a = numerator*other_rational.denominator + other_rational.numerator*denominator;
27     int b = denominator*other_rational.denominator;
28     rational sum(a,b); // Assign a new rational number
29     return sum; // Return this rational number
30 }
```

```
7     rational A(13,727); // Declare and initialise a rational number
8     rational B(231,983); // Declare and initialise a rational number
```

two rational numbers

```
13    rational C = A + B; // Can use + operation
```

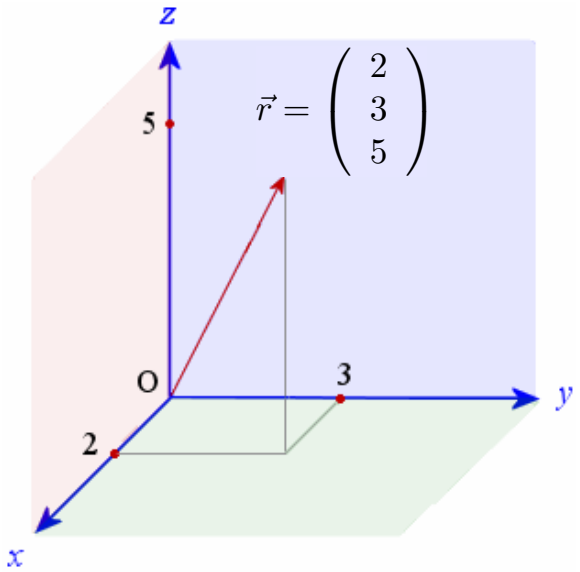
we can use the + operation just like any other number!

See workshop document: **structures_objects.pdf** for more info.

Building a 3d vector class

workshop_exercises.pdf

The workshop exercises invite you to explore a class constructed for vectors in 3d Euclidean space:



Recall a **vector** has both a length and a direction. Choosing cartesian axes it is specified by 3 components along each axis:

$$\vec{r} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

We stack components on top of each other to give a representation of a vector

We can view a vector as an arrow connecting the origin to the point specified.

Building a 3d vector class

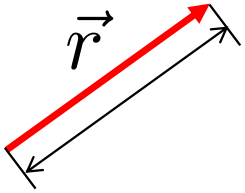
We can compute the overlap of two vectors using the **dot** product:

$$\vec{r} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \qquad \vec{u} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

$$\vec{r} \cdot \vec{u} = xa + yb + zc$$

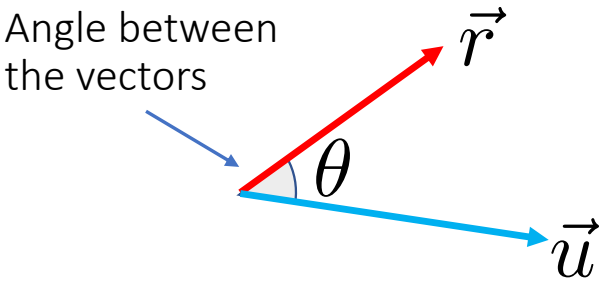
Takes **two vectors** and returns a **scalar**

The length of a vector is $|\vec{r}| = \sqrt{\vec{r} \cdot \vec{r}} = \sqrt{x^2 + y^2 + z^2}$



Geometrically we can interpret the dot product as:

$$\vec{r} \cdot \vec{u} = |\vec{r}||\vec{u}| \cos(\theta)$$



Orthogonal vectors

$$\vec{r} \cdot \vec{u} = 0$$

Building a 3d vector class

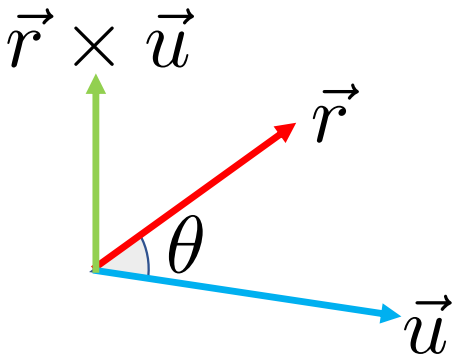
The **cross** product of two vectors is:

$$\vec{r} \times \vec{u} = \begin{pmatrix} yc - zb \\ za - xc \\ xb - ya \end{pmatrix}$$

Takes **two vectors** and returns a **vector**

Geometrically we can interpret the cross product as producing a new vector orthogonal to the first two with a magnitude:

$$|\vec{r} \times \vec{u}| = |\vec{r}||\vec{u}| \sin(\theta)$$



You are provided with a class definition for 3d vectors with basic operations like addition, negation and scalar multiplication are implemented. Your task is to add methods for computing a vector length, dot, cross product, and angle.