Numerical Computing in C++

Practical 3 (Lectures 5-6)

Section 1

This part of the practical covers the basics of writing a class.

1. Write an empty class called `Student`, and check it compiles
   a) Add public `std::string` member for name
   b) Add public `int` member for age
   c) Add public `double` member for height in metres
   d) Check that you can create objects of type `Student`, and that you can set their members
   e) Make each of those members private, and write public `set` and `get` methods for each
2. Create a vector with at least three students, and set their names, ages and heights
   a) Once per year you need to update each student's age! Do this with:
      - a standard `for` loop: `for(int i=0; i<students.size(); ++i)`
      - a range `for` loop: `for(auto& s:students)`
      - the `std::transform` algorithm
   b) You want to calculate the total height of all the students. Do this with:
      - a range `for` loop
      - the `std::accumulate` algorithm
3. Overload the default constructor so that it sets the name to `"not_set"`, the age to `-1` and the height to `0.0` by default
4. Create a new constructor so that you can construct `Student` directly with `name`
   - i.e. `Student s("Boris Johnson");`
5. Add default parameters to your new constructor so that age and height can be set during construction, but are given default values if not

Section 2

The final part of this practical covers writing a simple ODE integration stepper class. When implementing a solution to any given problem, it is important to think about how to break apart the problem into sub-problems to solve individually. For the case of ODE integration, you wish to solve the following problem:

$$\frac{dx}{dt} = f(t, x)$$

This naturally leads to two different sub-problems, (1) writing code to implement the rhs function $f(t, x)$, and (2) writing code to discretise the derivative $\frac{dx}{dt}$. We will use these sub-problems to inform the design of our ODE stepper class.

1. Write an ODE integration stepper class with the following properties:

   - A state variable `x` represented by a `double` type

- A internal timestep `dt` represented by a `double`
- A `dxdt` function that implements the rhs function $f(x) = -x$ (i.e. sub-problem (1) discussed above). It takes two arguments, a `const double` representing x and a reference to a `double` (i.e. `double&`) that returns $f(x) = -x$.
- A `step` function that takes a single integration step (i.e. sub-problem (2)) using an explicit Euler stepping scheme, and updates the internal state variable `x`.

2. Make the state variable and timestep variables `private`. Write one function that allows the user to set these private variables, and another function that can be used to access them (i.e. getter and setter functions). Only allow the timestep to be set to a positive and non-zero value.
3. Once you have a class that takes a single ODE integration step, the next problem to solve is how to integrate y over a given time range. Write an `integrate` function that takes an initial state `x0`, an integration time `t1`, and a step size `dt`. Use your stepper class to implement the function
4. Write test code which uses your integrate function to integrate $dx/dt = -x$ and verify it correctly calculates the solution $x(t) = exp(-t)$