UNIVERSITY OF BIRMINGHAM

University of Birmingham
Business School

MSc. Mathematical Finance

Assignment 2: Technical Report for Simulating N-Body Simulation Programme

1422827

System Requirement: By using two different discretization methods in order to produce results for 3 scenarios different scenarios. The application needs to be object orientated with a clear client interface and input validation. The N-body simulation application has been developed in C++ language conforming to the ISO C++11 standard. The development was done in the Dev-Cpp 5.11 environment using the GCC 4.9.2 compiler.

Feb 2018

Contents

1 Introduction

The purpose of this report is to describe an implementation of the N-body simulation application and assess its performance. This project involves multiple features, including object-oriented design, advanced polymorphic code, performance optimisation, and is written in a maintainable style. The report also includes a description of encountered problems with solutions and design decisions with justifications.

The N-body simulation is used to model the interaction of N astronomical bodies under the influence of gravity. It's common to consider a 2D version of the problem to simplify visualisation, but this project uses a three-dimensional implementation to achieve realistic results. An additional advantage of such a general-purpose approach is the ability to model different scenarios, including complicated cases with highly inclined orbits.

The next sections describe the main features of the application, the implementation details, the obtained results with assessment, and appendixes.

2 Features

2.1 Project Structure

The N-body simulation application has been developed in C++ language conforming to the ISO C++11 standard. The development was done in the Dev-Cpp 5.11 environment using the GCC 4.9.2 compiler.

The project consists of numerous files, with class declarations in header files and corresponding member function definitions in source files. To maintain a better structure, the source files are internally organised into folders in the Project Browser inside the Dev-Cpp environment.

2.2 Application Interface

Working with metres and seconds is inconvenient due to enormous values for distances and times. This complicates the output of simulation results and in general it's hard to interpret and compare large numbers. The application uses astronomical units instead of metres and years instead of seconds for the sake of convenience. Since the initial data is still in metres and seconds, the required conversion is handled by the Data Manager.

The application provides an intuitive client interface with input validation and an option to quit at any time. First, a simulation method and a scenario are selected from available choices. Then a number of years to simulate and a number of steps per year are entered. Finally, the option to export the detailed data to a file for each simulation step is specified. The results are displayed as a table of values before and after the simulation. At this stage, the user can try again or quit the application.

3 Implementation

The N-body simulation application is written using object-oriented programming techniques with polymorphic code and optimisations to run reasonably fast. The following subsections describe the critical parts of the project illustrating various cases of problem solving.

3.1 Simulation Methods

In order to seamlessly change different discretisations, the abstract base class MethodBase is introduced. It contains the pure virtual method Step() and cannot be instantiated directly. The polymorphic functionality is implemented in the derived classes MethodEuler and MethodRungeKutta. This is a highly extensible approach: it's possible to add other simulation methods using different implementations with the base class specifying a uniform interface.

3.2 Operator Overloading

Given N bodies, their instantaneous state can be represented as a vector of size 2N which contains their positions and velocities. This state is implemented in the StateVector class. This approach allows implementing the simulation formulas directly in vector notation. To conveniently update state vectors, this class overloads addition and multiplication operators. As a result, the Euler step is implemented in just one line of code and the Runge-Kutta step – in several lines only. In addition, one function is used for both methods to compute the derivative of the state vector: once in the Euler method and four times in Runge-Kutta method. Moreover, this computation function was optimised: there are no partial time increments because gravity doesn't explicitly depend on time[1].

3.3 Data Manager

The application uses more than 50 parameters, mostly initial data. For consistent usage throughout the application, each parameter is associated with a unique string. However, hard wiring these strings into the code would be error prone: typos cannot be caught at compile time. This project uses parameter classes to circumvent this problem. The Data Manager is implemented as a singleton class and provides a convenient interface to query values for each parameter. It has an additional responsibility to automatically convert metres to astronomical units for positions and metres per second to astronomical units per year for velocities.

3.4 Generalised Parameter Classes

Designed with extensibility in mind, the application allows adding more bodies for new scenarios which can lead to hundreds of parameters. Using so many parameter classes would be impractical, with harder maintenance outweighing their advantages. To solve this problem, a generalised approach is used to represent multiple parameters as combinations of base parameters. Instead of using a separate parameter for each value like "initial X position of the Earth", two constituent parameters are introduced: "Earth" and "PositionX". In addition to standard methods for querying the parameter value based on its name, the Data Manager provides additional methods to query the combined parameters based on two names. This way, the "Earth" parameter is shared with other Earth data like mass or velocity and the "PositionX" parameter is shared with other bodies like Venus or Mars. As a result, 51 values in data.txt are represented with only 16 parameter classes thanks to combining.

3.5 Table Output

To decouple output from simulation, the TableOutput object is implemented as a separate class with a registration interface. Displaying application output in a nicely formatted table is quite a challenge. To achieve this, the TableOutput class has registration methods to specify row/column locations in the table with corresponding label/value pairs. A std::map is a suitable container to store the registered entries: row/column pairs are used as map keys and strings are used as map values. When this map is later iterated during the actual output, its entries are automatically sorted first by row and then by column, as required for the table, thanks to the built-in comparison operator of std::pair.

4 Results

For output and export, the application uses astronomical units for positions and distances, years for time, and astronomical units per year for velocities and speeds. All results in this section use the Runge-Kutta method as more superior in terms of accuracy.

4.1 Simulation Results

To conveniently represent a system perturbed by a passing neutron star with altered initial velocity in a uniform manner, the fourth scenario was added to the project which is based on the third one. The initial velocity of the neutron star in the fourth scenario is aimed at the Sun more directly to explore its destructive effects on the planetary system.

A typical application output for scenario 2 (Sun-Venus-Earth-Mars-Jupiter) is shown in Figure 1. The simulation is performed for 100 years using 100,000 time steps per year with the Runge-Kutta method.

4.2 Exported Data

More sophisticated results are obtained using the export option. Positions from export.txt can be visualised by external applications. Figure 2 shows the three versions of the Venus, Earth, and Mars configuration: initial orbits, orbits after the neutron star passage, and orbits after the altered neutron star passage. These plots show how influential the neutron star can be for a planetary system depending on its initial velocity. However, using the export functionality increases the run time dramatically.

Figure 3 shows the evolution of Venus, Earth, and Mars distances from the Sun before and after the neutron star passage, which is shown as a vertical line. Oscillations on the right side indicate elliptic orbits after the passage, with Mars being affected the most.

Figure 4 is the same as Figure 3, but with the altered neutron star passage. This time there is more influence, with distances from the Sun oscillating stronger after the passage.

Figure 5 shows the evolution of Earth-Mars distance before and after the neutron star passage. Such a pattern is typical for periodic approaches to each other on concentric planetary orbits. The pattern becomes slightly perturbed on the right side.

Figure 6 is the same as Figure 5, but with the altered neutron star passage. This time the distance pattern on the right side becomes rather chaotic, reflecting the larger influence of the altered neutron star.

5 Assessment Review

The N-body simulation application was a great experience in object-oriented programming. With classes it's possible to break a complex application into separate objects responsible for their task which greatly simplifies the development. Abstract base classes define interfaces and derived classes specify different behaviours using polymorphism[2]. Overloaded operators provide a very elegant mechanism for writing concise and convenient code which closely resembles mathematical formulas being implemented.

With the intuitive client interface it's possible to analyse system stability and compare simulation accuracy of different methods. A convenient measure to look for is the Sun-Earth distance, which should remain 1 astronomical unit throughout the entire simulation process. Scenario 1 was simulated for 10000 years using 100 steps per year. With the Euler method, the final Sun-Earth distance is 101.1 astronomical units while with the Runge-Kutta method this distance is 0.9972 astronomical units. These values indicate that the Runge-Kutta is clearly superior in accuracy.

The same simulation was done for scenario 2. With the Euler method, the final Sun-Earth distance is 26.04 astronomical units while with the Runge-Kutta method this distance is 0.9978 astronomical units. Again, the Runge-Kutta method is much more accurate.

The runtime performance was measured with the Euler method being about 6 times faster. However, it requires orders of magnitude more time steps to achieve a similar accuracy as the Runge-Kutta method. This corresponds to theoretical estimations with errors decreasing according to the method order [3]. In addition to this, when comparing run times with other students, we found that, using Runge Kutta for scenario 3 (without using the functionality of export.txt) of 100 years, 100,000 time steps a year, my program took 4 minutes 57 seconds, the quickest of all 3 comparisons took 47 minutes and 28 seconds.

As a significantly more accurate option, the Runge-Kutta method was used for neutron star experiments. The visualisations in Figures 2-6 demonstrate the effect of a passing star on the planetary system. Scenario 3 represents a low influence situation with outer orbits slightly perturbed. Scenario 4 represents a more destructive case with orbits becoming elliptic and even intersecting each other. For example, the outer planet Mars now can be an inner planet at times.

With the extensible approach taken in this project, it's easy to make future additions to the application. A new simulation method can be added by implementing the defined interface of the abstract base class. New scenarios are also easy to add, as well as new parameter classes to use other astronomical bodies for simulation.

6 Appendix

```
Welcome to the N-body simulation application

Available simulation methods:
e) Euler discretization
r) Runge-Kutta discretization:

Method type (e, r, q - quit):  r

Available scenarios:
1) Sun-Earth
2) Sun-Venus-Earth-Mars-Jupiter
3) Sun-Venus-Earth-Mars perturbed by a passing neutron star
4) Sun-Venus-Earth-Mars perturbed by a passing neutron star (altered)

Scenario (1, 2, 3, 4, q - quit):  2

How many years to simulate? (q - quit):  10000

How many time steps per year? (q - quit):  100

Export to export.txt? (y, n, q - quit):  n
Simulating...
        Year    Var         Sun       Venus       Earth        Mars     Jupiter
           0  Pos X           0     -0.7233           1           0           0
              Pos Y           0           0           0       1.523      -5.204
              Pos Z           0     0.04285           0     0.04919     -0.1183
              Vel X           0           0           0      -5.065       2.757
              Vel Y           0      -7.387       6.282           0           0
              Vel Z           0           0           0           0           0
              Speed           0       7.387       6.282       5.065       2.757
              SDist           0      0.7245           1       1.524       5.205

     1e+004  Pos X       26.26       26.62       25.99       25.34       30.55
              Pos Y    0.005707     -0.5954      0.9641      -1.186      -2.957
              Pos Z -4.844e-005    -0.03661   -0.004947    -0.03976     -0.0673
              Vel X    0.001131       6.463      -6.039       4.083       1.571
              Vel Y   -0.002161       3.829      -1.757      -3.095       2.262
              Vel Z -4.879e-005     -0.2389      0.1056    -0.04511     0.05142
              Speed    0.002439       7.516        6.29       5.123       2.755
              SDist           0      0.6994      0.9978       1.508       5.208


Enter r to run again,  anything else to quit:
```

Figure 1: User inputs and simulation results for scenario 2.

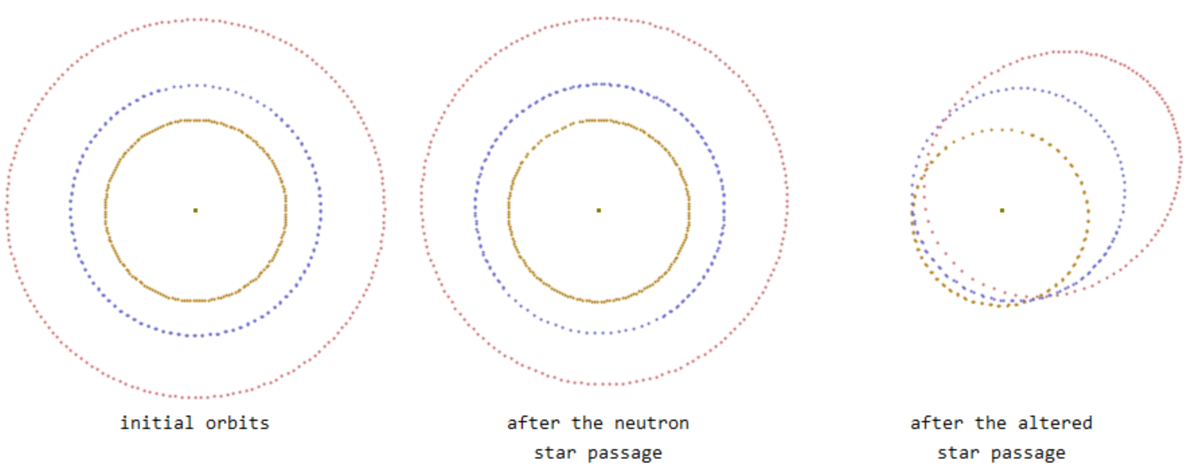

Figure 2: Venus, Earth, and Mars orbits: initial, scenario 3, scenario 4 (Runge-Kutta method, 100 time steps per year).

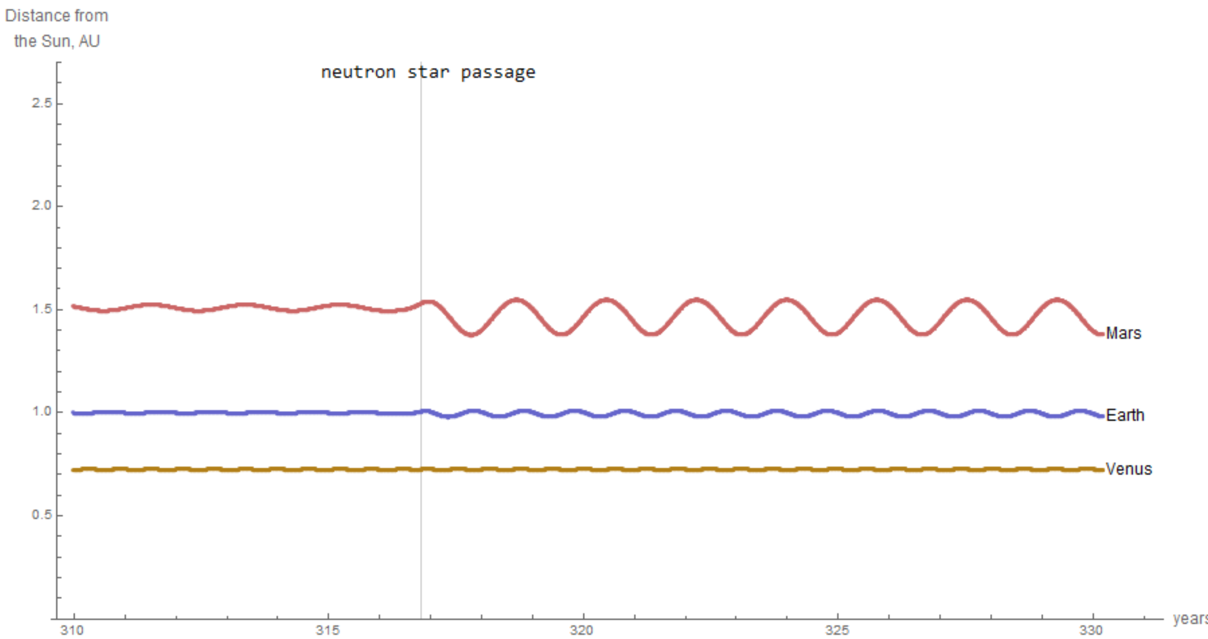

Figure 3: Venus, Earth, and Mars distances from the Sun, before and after the neutron star passage in scenario 3 (Runge-Kutta method, 100 time steps per year).

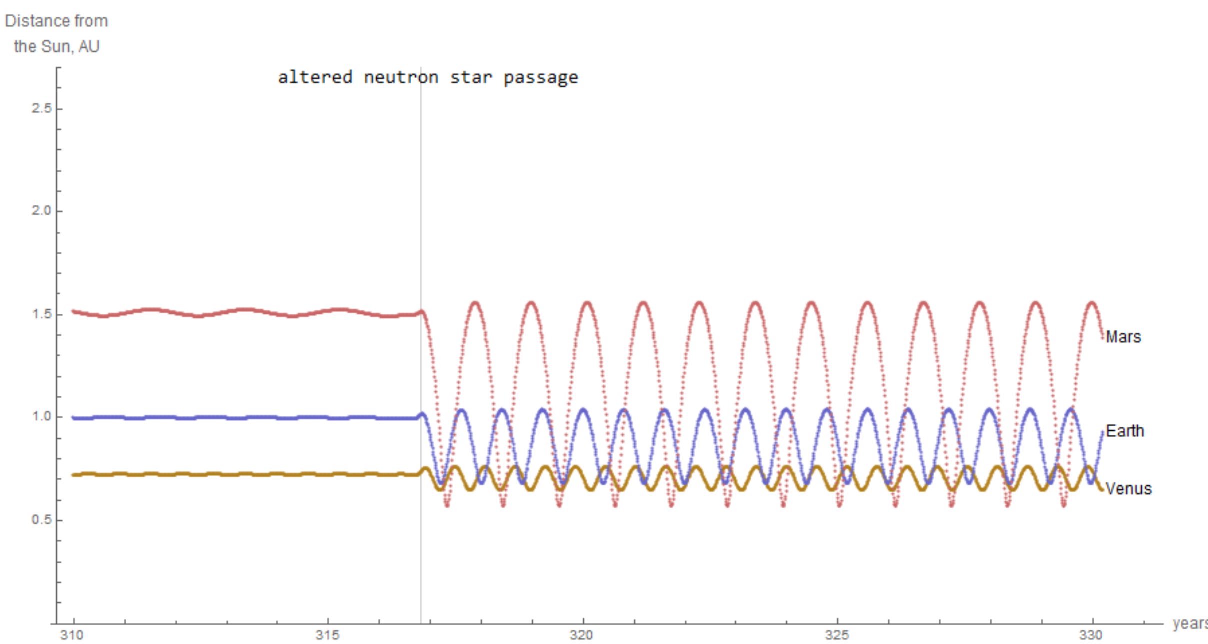

Figure 4: Venus, Earth, and Mars distances from the Sun, before and after the altered neutron star passage in scenario 4 (Runge-Kutta method, 100 time steps per year).

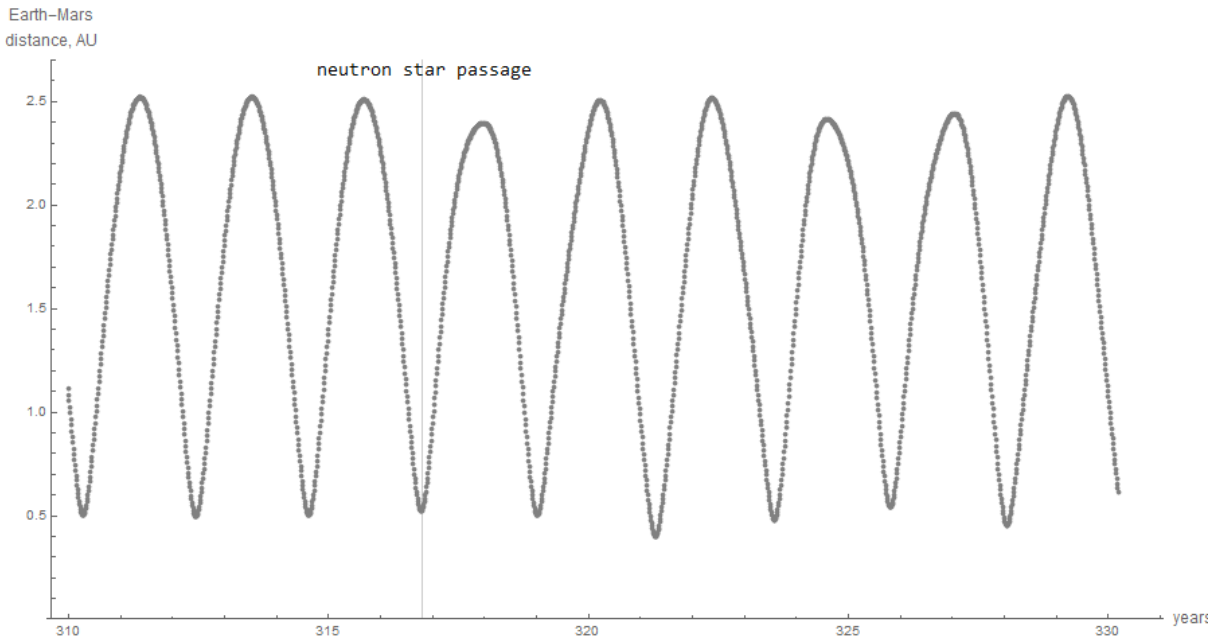

Figure 5: Earth-Mars distance evolution, before and after the neutron star passage in scenario 3 (Runge-Kutta method, 100 time steps per year).

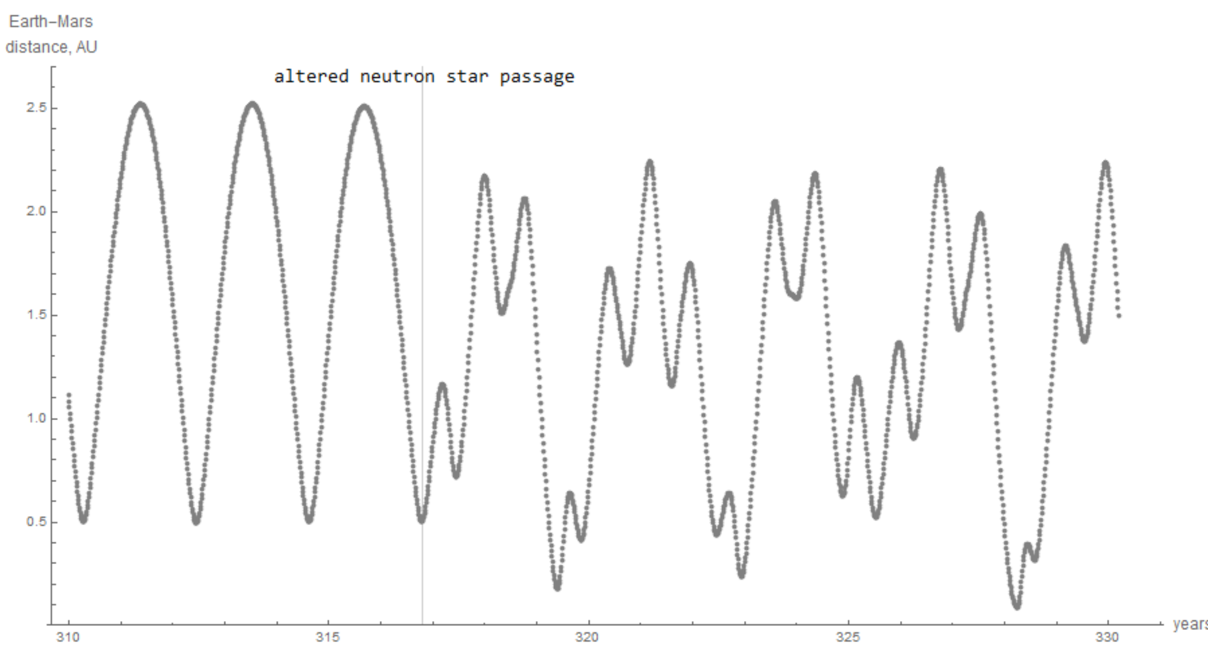

Figure 6: Earth-Mars distance evolution, before and after the altered neutron star passage in scenario 4 (Runge-Kutta method, 100 time steps per year).

7 References

[1] Shafiq Ur Rehman. Efficient algorithms for modelling close-encounters of the solar system. *Computer Integrated Manufacturing Systems*, (4):11 – 12, 2013.

[2] David Parsons. *Object oriented programming with C++*. Cengage Learning EMEA, 2000.

[3] John A Sokolowski and Catherine M Banks. *Modeling and simulation fundamentals: theoretical underpinnings and practical domains*. John Wiley & Sons, 2010.

8 Code Listing

Now moving onto the code listing, just as a justification, in order to prevent line wrap (or an overrun of the margins. I have edited so anything originally running over the 80 character space such as:

```
MethodBase * PseudoFactory::CreateMethod() const
{
    const char MethodType = inp_->GetMethodType();

    // Create the Method based on its type
    switch(MethodType)
    {
        case 'e': return new MethodEuler(*this); break;
        case 'r': return new MethodRungeKutta(*this); break;
        default: throw std::runtime_error("PseudoFactory::CreateMethod: Bad character");
    }
}
```

Then I have changed it, so it is now presented as:

```
MethodBase * PseudoFactory::CreateMethod() const
{
    const char MethodType = inp_->GetMethodType();

    // Create the Method based on its type
    switch(MethodType)
    {
        case 'e': return new MethodEuler(*this); break;
        case 'r': return new MethodRungeKutta(*this); break;
        default: throw std::runtime_error("PseudoFactory::CreateMethod:
                                                     Bad character");
    }
}
```

I believe this has successfully navigated the problems of line wrapping and I hope it is easy for one to understand

```cpp
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  main.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ApplicationWrapper.h"
#include "ErrorHandler.h"
#include "utility.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  main
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

int main(int argc, char *argv[])
{
    ErrorHandler handler;

    try
    {
        // Run the app repeatedly
        do
        {
            ApplicationWrapper app;
            app.Run();
        }
        while(ut::do_again());
    }
    catch(const std::runtime_error & e)
    {
        // Handle runtime errors
        handler.HandleRunTimeError(e);
    }
    catch(const std::exception&)
    {
        // Handle early exit
        return 0;
    }
    catch(...)
    {
        // Handle other errors
        handler.HandleUnknownError();
    }

    return 0;
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ApplicationBase.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ApplicationBase.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  functionality
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

// dummy

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ApplicationBase.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef ApplicationBaseH
#define ApplicationBaseH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class TableOutput;

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class ApplicationBase
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class ApplicationBase
{
    public:
        virtual ~ApplicationBase() {}
        virtual void RegisterOutput(TableOutput & out) const = 0;
        virtual void Run() = 0;
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```cpp
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  SimulationApplication.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "SimulationApplication.h"
#include "PseudoFactory.h"
#include "ScenarioBase.h"
#include "MethodBase.h"
#include "Export.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

SimulationApplication::SimulationApplication(const PseudoFactory & fac)
:   TimeStep_(fac.GetTimeStep())
,   NumSteps_(fac.GetNumSteps())
,   ExportFilename_(fac.GetExportFilename())
{
    scenario_ = fac.CreateScenario();
    method_ = fac.CreateMethod();
}

SimulationApplication::~SimulationApplication()
{
    delete scenario_;
    delete method_;
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Run
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

void SimulationApplication::Run()
{
    // If the filename was supplied, the export is needed
    const bool NeedExport = !ExportFilename_.empty();

    // Create an Export object if needed
    Export * exp = NeedExport ? new Export(ExportFilename_) : nullptr;

    // Create the start state and store it for later output
    StateVector state = scenario_->CreateY();
    StartState_ = state;

    // Simulate for a specified number steps
    for(long i = 0; i != NumSteps_; ++i)
    {
        // If export is needed
        if(exp)
        {
            // Register the state for export
            const double year = i*TimeStep_;
            scenario_->RegisterExport(*exp, state, year);
```

```
            // Write at once to avoid clogging memory with huge data
            const bool NeedHeader = (i == 0);
            exp->DoExport(NeedHeader);
        }

        // Use the specified method for a simulation step
        method_->Step(*scenario_, state);
    }

    // Store the end state for later output
    EndState_ = state;

    // Destroy the Export object (if created) to close its file
    delete exp;
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  RegisterOutput
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

void SimulationApplication::RegisterOutput(TableOutput & out) const
{
    // Start and end year
    const double StartYear = 0;
    const double EndYear = TimeStep_*NumSteps_;

    // Register start and end states
    long line = 0;
    scenario_->RegisterOutput(out, line, StartState_, StartYear);
    scenario_->RegisterOutput(out, line, EndState_, EndYear);
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  SimulationApplication.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef SimulationApplicationH
#define SimulationApplicationH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ApplicationBase.h"
#include "StateVector.h"
#include <string>

class PseudoFactory;
class ScenarioBase;
class MethodBase;

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class SimulationApplication
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class SimulationApplication : public ApplicationBase
{
    public:
        SimulationApplication(const PseudoFactory & fac);
        ~SimulationApplication();

        void Run();
        void RegisterOutput(TableOutput & out) const;

    private:
        ScenarioBase * scenario_;
        MethodBase * method_;

        double TimeStep_;               // Simulation time step
        long NumSteps_;                 // Number of simulation steps

        std::string ExportFilename_;    // Filename for exporting

        StateVector StartState_;        // State before simulation
        StateVector EndState_;          // State after simulation
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ApplicationWrapper.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ApplicationWrapper.h"
#include "PseudoFactory.h"
#include "Input.h"
#include "TableOutput.h"
#include "ApplicationBase.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ApplicationWrapper::ApplicationWrapper()
:   inp_(nullptr)
,   out_(nullptr)
,   fac_(nullptr)
,   app_(nullptr)
{
    // Create the input object
    inp_ = new Input;

    // If early exit is requested
    if(inp_->GetQuitOption())
    {
        // Exit cleanly
        delete inp_;
        inp_ = nullptr;
        throw std::exception();
    }

    // Create the output and factory objects
    out_ = new TableOutput;
    fac_ = new PseudoFactory;

    // Set input and output
    fac_->SetInput(inp_);
    fac_->SetOutput(out_);

    // Create the application object
    app_ = fac_->CreateApplication();
}

ApplicationWrapper::~ApplicationWrapper()
{
    // Free the contained objects
    delete fac_;
    delete inp_;
    delete out_;
    delete app_;
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//  Run
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

void ApplicationWrapper::Run() const
{
    // Perform the simulation
    out_->OutputBanner("Simulating...");
    app_->Run();

    // Output the results
    app_->RegisterOutput(*out_);
    out_->DoOutput();
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ApplicationWrapper.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef ApplicationWrapperH
#define ApplicationWrapperH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class PseudoFactory;
class Input;
class TableOutput;
class ApplicationBase;

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class ApplicationWrapper
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class ApplicationWrapper
{
    public:
        ApplicationWrapper();
        ~ApplicationWrapper();

        void Run() const;

    private:
        Input * inp_;
        TableOutput * out_;
        PseudoFactory * fac_;
        ApplicationBase * app_;
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ErrorHandler.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ErrorHandler.h"
#include "utility.h"
#include <iostream>

void ErrorHandler::HandleRunTimeError(const std::runtime_error & e) const
{
    std::cout << "Error caught:  " << e.what() << std::endl;
}

void ErrorHandler::HandleUnknownError() const
{
    std::cout << "Unknown error caught" << std::endl;
}

long ErrorHandler::PauseAndReturn() const
{
    return ut::PauseAndReturn();
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ErrorHandler.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef ErrorHandlerH
#define ErrorHandlerH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include <stdexcept>

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class ErrorHandler
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class ErrorHandler
{
    public:
        void HandleRunTimeError(const std::runtime_error & e) const;
        void HandleUnknownError() const;
        long PauseAndReturn() const;
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  StateVector.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "StateVector.h"
#include <cmath>

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Resizing
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

void StateVector::Resize(long NumBodies)
{
    // Set the number of bodies
    NumBodies_ = NumBodies;

    // Number of bodies doubled for positions and velocities
    states_.resize(NumBodies*2);
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Setters and getters
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

void StateVector::SetPosition(long i, const std::vector<double> & P)
{
    states_[i] = P;
}

std::vector<double> StateVector::GetPosition(long i) const
{
    return states_[i];
}

void StateVector::SetVelocity(long i, const std::vector<double> & V)
{
    states_[NumBodies_ + i] = V;
}

std::vector<double> StateVector::GetVelocity(long i) const
{
    return states_[NumBodies_ + i];
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Computed values
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

double StateVector::ComputeSpeed(long i) const
{
    // Get the body velocity
    const std::vector<double> v = GetVelocity(i);

    // Velocity vector magnitude
```

```
    return std::sqrt(v[0]*v[0] + v[1]*v[1] + v[2]*v[2]);
}

double StateVector::ComputeSunDistance(long i) const
{
    // Get the body and Sun positions
    const std::vector<double> bp = GetPosition(i);
    const std::vector<double> sp = GetPosition(0);

    // Relative vector
    const double dx = bp[0] - sp[0];
    const double dy = bp[1] - sp[1];
    const double dz = bp[2] - sp[2];

    // Relative vector magnitude
    return std::sqrt(dx*dx + dy*dy + dz*dz);
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Overloaded operators
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

StateVector operator+(const StateVector & a, const StateVector & b)
{
    // Use the overloaded operation+=
    StateVector temp(a);
    temp += b;
    return temp;
}

StateVector operator*(double a, const StateVector & b)
{
    // Use the overloaded operation*=
    StateVector temp(b);
    temp *= a;
    return temp;
}

StateVector & StateVector::operator+=(const StateVector & a)
{
    // For each state
    for(long i = 0; i != NumBodies_*2; ++i)
    {
        // add X, Y, Z components
        states_[i][0] += a.states_[i][0];
        states_[i][1] += a.states_[i][1];
        states_[i][2] += a.states_[i][2];
    }

    return *this;
}

StateVector & StateVector::operator*=(double a)
{
    // For each state
```

```
    for(long i = 0; i != NumBodies_*2; ++i)
    {
        // scale X, Y, Z components
        states_[i][0] *= a;
        states_[i][1] *= a;
        states_[i][2] *= a;
    }

    return *this;
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  StateVector.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef StateVectorH
#define StateVectorH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include <vector>

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class StateVector
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class StateVector
{
    public:
        // Rule of three not required by this class, defaults work fine
        StateVector() : NumBodies_(0) {}

        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        //  Resizing
        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

        void Resize(long NumBodies);

        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        //  Setters and getters
        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

        void SetPosition(long i, const std::vector<double> & P);
        std::vector<double> GetPosition(long i) const;
        void SetVelocity(long i, const std::vector<double> & V);
        std::vector<double> GetVelocity(long i) const;

        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        //  Computed values
        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

        double ComputeSpeed(long i) const;
        double ComputeSunDistance(long i) const;

        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        //  Overloaded operators
        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

        friend StateVector operator+(const StateVector & a, const StateVector
                                                                        & b);
        friend StateVector operator*(double a, const StateVector & b);

        StateVector & operator+=(const StateVector & a);
        StateVector & operator*=(double a);

    private:
        long NumBodies_;                    // Number of bodies
```

```
        typedef std::vector<double> State;  // 3D vector for position/velocity
        std::vector<State> states_;         // State array
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  MethodBase.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "MethodBase.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  functionality
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

// dummy

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  MethodBase.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef MethodBaseH
#define MethodBaseH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class ScenarioBase;
class StateVector;

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class MethodBase
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class MethodBase
{
    public:
        virtual ~MethodBase() {}
        virtual void Step(const ScenarioBase & scenario, StateVector & y) = 0;
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  MethodEuler.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "MethodEuler.h"
#include "PseudoFactory.h"
#include "StateVector.h"
#include "ScenarioBase.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

MethodEuler::MethodEuler(const PseudoFactory & fac)
:   h_(fac.GetTimeStep())
{
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Step
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

void MethodEuler::Step(const ScenarioBase & scenario, StateVector & y)
{
    // Euler discretization
    y += h_*scenario.ComputeF(y);
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  MethodEuler.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef MethodEulerH
#define MethodEulerH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "MethodBase.h"

class PseudoFactory;

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class MethodEuler
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class MethodEuler : public MethodBase
{
    public:
        MethodEuler(const PseudoFactory & fac);
        void Step(const ScenarioBase & scenario, StateVector & y);

    private:
        double h_;   // Simulation time step
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  MethodRungeKutta.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "MethodRungeKutta.h"
#include "PseudoFactory.h"
#include "StateVector.h"
#include "ScenarioBase.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

MethodRungeKutta::MethodRungeKutta(const PseudoFactory & fac)
:   h_(fac.GetTimeStep())
{
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Step
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

void MethodRungeKutta::Step(const ScenarioBase & scenario, StateVector & y)
{
    // Constants
    static const double c6 = 1.0/6;
    static const double c3 = 1.0/3;

    // Runge-Kutta discretization
    // there are no partial time increments
    // because gravity doesn't explicitly depend on time
    const StateVector d1 = h_*scenario.ComputeF(y);
    const StateVector d2 = h_*scenario.ComputeF(y + 0.5*d1);
    const StateVector d3 = h_*scenario.ComputeF(y + 0.5*d2);
    const StateVector d4 = h_*scenario.ComputeF(y + d3);
    y += c6*d1 + c3*d2 + c3*d3 + c6*d4;
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  MethodRungeKutta.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef MethodRungeKuttaH
#define MethodRungeKuttaH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "MethodBase.h"

class PseudoFactory;

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class MethodRungeKutta
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class MethodRungeKutta : public MethodBase
{
    public:
        MethodRungeKutta(const PseudoFactory & fac);
        void Step(const ScenarioBase & scenario, StateVector & y);

    private:
        double h_;   // Simulation time step
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ScenarioBase.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ScenarioBase.h"
#include "StateVector.h"
#include "ParameterG.h"
#include "DataManager.h"
#include "TableOutput.h"
#include "Export.h"
#include <cmath>

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Adding bodies to the scenario
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

void ScenarioBase::AddBody(const std::string & name)
{
    // Add the body name
    names_.push_back(name);

    // Get and add the body mass
    const double mass = DataManager::Instance().GetMass(name);
    masses_.push_back(mass);

    // Update the number of bodies
    ++NumBodies_;
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Initial state
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

StateVector ScenarioBase::CreateY() const
{
    // Create and resize the state vector
    StateVector y;
    y.Resize(NumBodies_);

    // For each body
    for(long i = 0; i != NumBodies_; ++i)
    {
        // Get the initial position and velocity
        const std::vector<double> p = DataManager::Instance().GetInitialPosition
                                                                (names_[i]);
        const std::vector<double> v = DataManager::Instance().GetInitialVelocity
                                                                (names_[i]);

        // Set the position and velocity in the state vector
        y.SetPosition(i, p);
        y.SetVelocity(i, v);
    }

    return y;
```

```
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  State vector derivative function
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

StateVector ScenarioBase::ComputeF(const StateVector & y) const
{
    // Create and resize the state vector
    StateVector f;
    f.Resize(NumBodies_);

    // For each body
    for(long i = 0; i != NumBodies_; ++i)
    {
        // Get the velocity and compute the acceleration
        const std::vector<double> v = y.GetVelocity(i);
        const std::vector<double> a = ComputeAcceleration(y, i);

        // Set the velocity and acceleration in the state vector
        f.SetPosition(i, v);
        f.SetVelocity(i, a);
    }

    return f;
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Acceleration due to gravity acting on body i
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

std::vector<double> ScenarioBase::ComputeAcceleration(const StateVector & y,
                                                                long i) const
{
    // Gravitational constant
    static const double G = DataManager::Instance().GetValue(ParameterG().name);

    // Position of body i, cached for performance
    const std::vector<double> Pi = y.GetPosition(i);
    const double xi = Pi[0];
    const double yi = Pi[1];
    const double zi = Pi[2];

    // Acceleration components, initially zero
    double ax = 0;
    double ay = 0;
    double az = 0;

    // For each body
    for(long k = 0; k != NumBodies_; ++k)
    {
        // Ignore itself
        if(k != i)
        {
            // Position of body k
```

```
            const std::vector<double> Pk = y.GetPosition(k);

            // Direction vector from body i to body k
            const double dx = Pk[0] - xi;
            const double dy = Pk[1] - yi;
            const double dz = Pk[2] - zi;

            // Distance between the bodies
            const double d = std::sqrt(dx*dx + dy*dy + dz*dz);

            // Acceleration formula
            const double c = masses_[k]/(d*d*d);
            ax += c*dx;
            ay += c*dy;
            az += c*dz;
        }
    }

    // Acceleration multiplied by G outside the loop for performance
    return std::vector<double>({G*ax, G*ay, G*az});
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  RegisterOutput
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

void ScenarioBase::RegisterOutput(TableOutput & out, long & line,
                                  const StateVector & state, double year) const
{
    // Register the table header
    if(year == 0.0)
    {
        // Register the Year and Variable text
        out.RegisterOutput(line, 0, "Year");
        out.RegisterOutput(line, 1, "Var");

        // Register the body names
        for(long i = 0; i != NumBodies_; ++i)
            out.RegisterOutput(line, 2 + i, names_[i]);
    }

    // Register the Year value
    out.RegisterOutput(line + 1, 0, year);

    // Register empty strings for the remaining Year column
    for(long row = line + 2; row != line + 9; ++row)
        out.RegisterOutput(row, 0, "");

    // For each body
    for(long i = 0; i != NumBodies_; ++i)
    {
        // Get the body position, velocity, speed and distance from the Sun
        const std::vector<double> p = state.GetPosition(i);
        const std::vector<double> v = state.GetVelocity(i);
        const double speed = state.ComputeSpeed(i);
```

```
        const double distance = state.ComputeSunDistance(i);

        // Register the body values at consecutive rows
        long row = line;
        out.RegisterOutput(++row, 1, "Pos X", 2 + i, p[0]);
        out.RegisterOutput(++row, 1, "Pos Y", 2 + i, p[1]);
        out.RegisterOutput(++row, 1, "Pos Z", 2 + i, p[2]);
        out.RegisterOutput(++row, 1, "Vel X", 2 + i, v[0]);
        out.RegisterOutput(++row, 1, "Vel Y", 2 + i, v[1]);
        out.RegisterOutput(++row, 1, "Vel Z", 2 + i, v[2]);
        out.RegisterOutput(++row, 1, "Speed", 2 + i, speed);
        out.RegisterOutput(++row, 1, "SDist", 2 + i, distance);
    }

    // Register an empty line at the end
    out.RegisterOutput(line + 9, 0, "");

    // Update the output line
    line += 10;
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  RegisterExport
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

void ScenarioBase::RegisterExport(Export & exp, const StateVector & state,
                                  double year) const
{
    // Register the year
    long column = 0;
    exp.RegisterExport(column, "Year", year);

    // For each body
    for(long i = 0; i != NumBodies_; ++i)
    {
        // Get the body name and position
        const std::string name = names_[i];
        const std::vector<double> pos = state.GetPosition(i);

        // Register the position components at consecutive columns
        exp.RegisterExport(++column, name + " Pos X", pos[0]);
        exp.RegisterExport(++column, name + " Pos Y", pos[1]);
        exp.RegisterExport(++column, name + " Pos Z", pos[2]);
    }
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ScenarioBase.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef ScenarioBaseH
#define ScenarioBaseH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include <string>
#include <vector>

class StateVector;
class TableOutput;
class Export;

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class ScenarioBase
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class ScenarioBase
{
    public:
        ScenarioBase() : NumBodies_(0) {}
        virtual ~ScenarioBase() {}

        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        //  Adding bodies to the scenario
        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

        void AddBody(const std::string & name);

        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        //  Initial state
        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

        StateVector CreateY() const;

        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        //  State vector derivative function
        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

        StateVector ComputeF(const StateVector & y) const;

        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        //  Output and Export Registration
        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

        void RegisterOutput(TableOutput & out, long & line,
                            const StateVector & state, double year) const;

        void RegisterExport(Export & exp, const StateVector & state,
                            double year) const;

private:
        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
        //  Acceleration due to gravity acting on body i
        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

        std::vector<double> ComputeAcceleration(const StateVector & y, long i)
                                                                        const;

        long NumBodies_;                    // Number of bodies
        std::vector<std::string> names_;    // Body names
        std::vector<double> masses_;        // Body masses
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Scenario1.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "Scenario1.h"
#include "ParameterSun.h"
#include "ParameterEarth.h"
#include "PseudoFactory.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Scenario1::Scenario1(const PseudoFactory & fac)
{
    // System containing only Sun-Earth
    AddBody(ParameterSun().name);
    AddBody(ParameterEarth().name);
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Scenario1.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef Scenario1H
#define Scenario1H
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ScenarioBase.h"

class PseudoFactory;

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class Scenario1
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class Scenario1 : public ScenarioBase
{
    public:
        Scenario1(const PseudoFactory & fac);
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Scenario2.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "Scenario2.h"
#include "ParameterSun.h"
#include "ParameterVenus.h"
#include "ParameterEarth.h"
#include "ParameterMars.h"
#include "ParameterJupiter.h"
#include "PseudoFactory.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Scenario2::Scenario2(const PseudoFactory & fac)
{
    // System containing only Sun-Venus-Earth-Mars-Jupiter
    AddBody(ParameterSun().name);
    AddBody(ParameterVenus().name);
    AddBody(ParameterEarth().name);
    AddBody(ParameterMars().name);
    AddBody(ParameterJupiter().name);
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Scenario2.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef Scenario2H
#define Scenario2H
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ScenarioBase.h"

class PseudoFactory;

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class Scenario2
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class Scenario2 : public ScenarioBase
{
    public:
        Scenario2(const PseudoFactory & fac);
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Scenario3.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "Scenario3.h"
#include "ParameterSun.h"
#include "ParameterVenus.h"
#include "ParameterEarth.h"
#include "ParameterMars.h"
#include "ParameterNeutronStar.h"
#include "PseudoFactory.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Scenario3::Scenario3(const PseudoFactory & fac)
{
    // System containing only Sun-Venus-Earth-Mars
    // perturbed by a passing neutron star
    AddBody(ParameterSun().name);
    AddBody(ParameterVenus().name);
    AddBody(ParameterEarth().name);
    AddBody(ParameterMars().name);
    AddBody(ParameterNeutronStar().name);
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Scenario3.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef Scenario3H
#define Scenario3H
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ScenarioBase.h"

class PseudoFactory;

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class Scenario3
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class Scenario3 : public ScenarioBase
{
    public:
        Scenario3(const PseudoFactory & fac);
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Scenario4.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "Scenario4.h"
#include "ParameterSun.h"
#include "ParameterVenus.h"
#include "ParameterEarth.h"
#include "ParameterMars.h"
#include "ParameterAlteredStar.h"
#include "PseudoFactory.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Scenario4::Scenario4(const PseudoFactory & fac)
{
    // System containing only Sun-Venus-Earth-Mars
    // perturbed by a passing neutron star
    AddBody(ParameterSun().name);
    AddBody(ParameterVenus().name);
    AddBody(ParameterEarth().name);
    AddBody(ParameterMars().name);
    AddBody(ParameterAlteredStar().name);
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Scenario4.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef Scenario4H
#define Scenario4H
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ScenarioBase.h"

class PseudoFactory;

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class Scenario4
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class Scenario4 : public ScenarioBase
{
    public:
        Scenario4(const PseudoFactory & fac);
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  PseudoFactory.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "PseudoFactory.h"

#include "Input.h"

#include "Scenario2.h"
#include "Scenario1.h"
#include "Scenario3.h"
#include "Scenario4.h"

#include "MethodEuler.h"
#include "MethodRungeKutta.h"

#include "SimulationApplication.h"

#include <stdexcept>

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Getters
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

double PseudoFactory::GetTimeStep() const
{
    return inp_->GetTimeStep();
}

long PseudoFactory::GetNumSteps() const
{
    return inp_->GetNumSteps();
}

std::string PseudoFactory::GetExportFilename() const
{
    return inp_->GetExportFilename();
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  CreateScenario
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ScenarioBase * PseudoFactory::CreateScenario() const
{
    const char ScenarioNumber = inp_->GetScenarioNumber();

    // Create the Scenario based on its number
    switch(ScenarioNumber)
    {
        case '1': return new Scenario1(*this); break;
        case '2': return new Scenario2(*this); break;
        case '3': return new Scenario3(*this); break;
        case '4': return new Scenario4(*this); break;
```

```
        default: throw std::runtime_error("PseudoFactory::CreateScenario:
                                                         Bad character");
    }
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  CreateMethod
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

MethodBase * PseudoFactory::CreateMethod() const
{
    const char MethodType = inp_->GetMethodType();

    // Create the Method based on its type
    switch(MethodType)
    {
        case 'e': return new MethodEuler(*this); break;
        case 'r': return new MethodRungeKutta(*this); break;
        default: throw std::runtime_error("PseudoFactory::CreateMethod:
                                                             Bad character");
    }
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  CreateApplication
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ApplicationBase * PseudoFactory::CreateApplication() const
{
    return new SimulationApplication(*this);
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  PseudoFactory.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef PseudoFactoryH
#define PseudoFactoryH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include <string>

class ScenarioBase;
class MethodBase;
class ApplicationBase;

class Input;
class TableOutput;

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class PseudoFactory
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class PseudoFactory
{
    public:
        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        //  Getters and setters
        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

        double GetTimeStep() const;
        long GetNumSteps() const;
        std::string GetExportFilename() const;

        void SetInput(Input * inp) {inp_ = inp;}
        void SetOutput(TableOutput * out) {out_ = out;}

        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        //  Creation methods
        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

        ScenarioBase * CreateScenario() const;
        MethodBase * CreateMethod() const;
        ApplicationBase * CreateApplication() const;

    private:
        Input * inp_;
        TableOutput * out_;
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterBase.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterBase.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  functionality
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

// dummy

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterBase.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef ParameterBaseH
#define ParameterBaseH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include <string>

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class ParameterBase
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class ParameterBase
{
    public:
        virtual ~ParameterBase() {}

        std::string name;           // Parameter name
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterAU.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterAU.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ParameterAU::ParameterAU()
{
    name = "AU";
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterAU.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef ParameterAUH
#define ParameterAUH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterBase.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class ParameterAU
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class ParameterAU : public ParameterBase
{
    public:
        ParameterAU();
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterPositionX.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterPositionX.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ParameterPositionX::ParameterPositionX()
{
    name = "PositionX";
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterPositionX.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef ParameterPositionXH
#define ParameterPositionXH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterBase.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class ParameterPositionX
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class ParameterPositionX : public ParameterBase
{
    public:
        ParameterPositionX();
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterPositionY.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterPositionY.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ParameterPositionY::ParameterPositionY()
{
    name = "PositionY";
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterPositionY.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef ParameterPositionYH
#define ParameterPositionYH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterBase.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class ParameterPositionY
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class ParameterPositionY : public ParameterBase
{
    public:
        ParameterPositionY();
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterPositionZ.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterPositionZ.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ParameterPositionZ::ParameterPositionZ()
{
    name = "PositionZ";
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterPositionZ.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef ParameterPositionZH
#define ParameterPositionZH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterBase.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class ParameterPositionZ
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class ParameterPositionZ : public ParameterBase
{
    public:
        ParameterPositionZ();
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterVelocityX.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterVelocityX.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ParameterVelocityX::ParameterVelocityX()
{
    name = "VelocityX";
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterVelocityX.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef ParameterVelocityXH
#define ParameterVelocityXH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterBase.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class ParameterVelocityX
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class ParameterVelocityX : public ParameterBase
{
    public:
        ParameterVelocityX();
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterVelocityY.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterVelocityY.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ParameterVelocityY::ParameterVelocityY()
{
    name = "VelocityY";
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterVelocityY.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef ParameterVelocityYH
#define ParameterVelocityYH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterBase.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class ParameterVelocityY
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class ParameterVelocityY : public ParameterBase
{
    public:
        ParameterVelocityY();
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterVelocityZ.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterVelocityZ.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ParameterVelocityZ::ParameterVelocityZ()
{
    name = "VelocityZ";
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterVelocityZ.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef ParameterVelocityZH
#define ParameterVelocityZH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterBase.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class ParameterVelocityZ
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class ParameterVelocityZ : public ParameterBase
{
    public:
        ParameterVelocityZ();
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterMass.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterMass.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ParameterMass::ParameterMass()
{
    name = "Mass";
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterMass.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef ParameterMassH
#define ParameterMassH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterBase.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class ParameterMass
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class ParameterMass : public ParameterBase
{
    public:
        ParameterMass();
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterG.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterG.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ParameterG::ParameterG()
{
    name = "G";
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterG.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef ParameterGH
#define ParameterGH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterBase.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class ParameterG
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class ParameterG : public ParameterBase
{
    public:
        ParameterG();
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterSun.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterSun.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ParameterSun::ParameterSun()
{
    name = "Sun";
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterSun.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef ParameterSunH
#define ParameterSunH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterBase.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class ParameterSun
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class ParameterSun : public ParameterBase
{
    public:
        ParameterSun();
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterEarth.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterEarth.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ParameterEarth::ParameterEarth()
{
    name = "Earth";
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterEarth.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef ParameterEarthH
#define ParameterEarthH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterBase.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class ParameterEarth
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class ParameterEarth : public ParameterBase
{
    public:
        ParameterEarth();
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterJupiter.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterJupiter.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ParameterJupiter::ParameterJupiter()
{
    name = "Jupiter";
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterJupiter.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef ParameterJupiterH
#define ParameterJupiterH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterBase.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class ParameterJupiter
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class ParameterJupiter : public ParameterBase
{
    public:
        ParameterJupiter();
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterMars.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterMars.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ParameterMars::ParameterMars()
{
    name = "Mars";
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterMars.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef ParameterMarsH
#define ParameterMarsH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterBase.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class ParameterMars
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class ParameterMars : public ParameterBase
{
    public:
        ParameterMars();
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterNeutronStar.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterNeutronStar.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ParameterNeutronStar::ParameterNeutronStar()
{
    name = "NeutronStar";
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterNeutronStar.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef ParameterNeutronStarH
#define ParameterNeutronStarH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterBase.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class ParameterNeutronStar
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class ParameterNeutronStar : public ParameterBase
{
    public:
        ParameterNeutronStar();
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterVenus.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterVenus.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ParameterVenus::ParameterVenus()
{
    name = "Venus";
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterVenus.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef ParameterVenusH
#define ParameterVenusH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterBase.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class ParameterVenus
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class ParameterVenus : public ParameterBase
{
    public:
        ParameterVenus();
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterAlteredStar.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterAlteredStar.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

ParameterAlteredStar::ParameterAlteredStar()
{
    name = "AlteredStar";
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  ParameterAlteredStar.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef ParameterAlteredStarH
#define ParameterAlteredStarH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "ParameterBase.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class ParameterAlteredStar
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class ParameterAlteredStar : public ParameterBase
{
    public:
        ParameterAlteredStar();
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  DataManager.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "DataManager.h"
#include "ParameterAU.h"
#include "ParameterMass.h"
#include "ParameterPositionX.h"
#include "ParameterPositionY.h"
#include "ParameterPositionZ.h"
#include "ParameterVelocityX.h"
#include "ParameterVelocityY.h"
#include "ParameterVelocityZ.h"
#include <fstream>
#include <stdexcept>

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

DataManager::DataManager()
{
    // Open the data file with error checking
    std::ifstream stream("data.txt");
    if(!stream.is_open())
    {
        throw std::runtime_error("Cannot open file data.txt");
    }

    // Populate the map with the file data
    while(!stream.eof())
    {
        // Read the name-value pair
        std::pair<std::string, double> data;
        stream >> data.first;
        stream >> data.second;

        // Insert it into the map
        if(stream.good()) map_.insert(data);
    }
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Getters
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

double DataManager::GetValue(const std::string & name) const
{
    // Get the value for the specified name
    const MapIt p = map_.find(name);

    // Error checking
    if(p == map_.end())
        throw std::runtime_error(name + " parameter not found");
```

```
    // Return the corresponding value
    return p->second;
}

double DataManager::GetValue(const std::string & name, const std::string &
                                                                  subname) const
{
    // Combine the name with subname to support parameter names like "JupiterMass"
    return GetValue(name + subname);
}

double DataManager::GetMass(const std::string & name) const
{
    // Get the mass for the specified body name
    return GetValue(name, ParameterMass().name);
}

std::vector<double> DataManager::GetInitialPosition(const std::string & name) const
{
    // Get the initial position for the specified body name
    const double x = GetValue(name, ParameterPositionX().name);
    const double y = GetValue(name, ParameterPositionY().name);
    const double z = GetValue(name, ParameterPositionZ().name);

    // Get the astronomical unit
    static const double AU = GetValue(ParameterAU().name);

    // Convert metres to AU units
    std::vector<double> p(3);
    p[0] = x/AU;
    p[1] = y/AU;
    p[2] = z/AU;
    return p;
}

std::vector<double> DataManager::GetInitialVelocity(const std::string & name) const
{
    // Get the initial velocity for the specified body name
    const double x = GetValue(name, ParameterVelocityX().name);
    const double y = GetValue(name, ParameterVelocityY().name);
    const double z = GetValue(name, ParameterVelocityZ().name);

    // Get the astronomical unit
    static const double AU = GetValue(ParameterAU().name);
    static const double SecondsInYear = 3600*24*365.25;

    // Convert metres/second to AU/year
    std::vector<double> v(3);
    v[0] = x*SecondsInYear/AU;
    v[1] = y*SecondsInYear/AU;
    v[2] = z*SecondsInYear/AU;
    return v;
}
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  DataManager.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef DataManagerH
#define DataManagerH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include <string>
#include <vector>
#include <map>

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class DataManager
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class DataManager
{
    public:

        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        //  Singleton access
        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

        static DataManager & Instance()
        {
            static DataManager manager;
            return manager;
        }

        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        //  Getters
        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

        double GetValue(const std::string & name) const;
        double GetValue(const std::string & name, const std::string & subname)
                                                                         const;
        double GetMass(const std::string & name) const;
        std::vector<double> GetInitialPosition(const std::string & name) const;
        std::vector<double> GetInitialVelocity(const std::string & name) const;

    private:
        DataManager();  // disable object creation for singleton

        typedef std::map<std::string, double> MapType;  // Map type
        typedef MapType::const_iterator MapIt;          // Map iterator
        MapType map_;                                   // Name-value pairs
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Input.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "Input.h"
#include "utility.h"
#include "lib_val.h"

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Input::Input()
:   QuitOption_(false)
{
    // Input the method type
    MethodType_ = InputMethodType();
    if(QuitOption_) return;

    // Input scenario number
    ScenarioNumber_ = InputScenarioNumber();
    if(QuitOption_) return;

    // Input the number of years
    double Years = InputYears();
    if(QuitOption_) return;

    // Input the steps per year
    long StepsPerYear = InputStepsPerYear();
    if(QuitOption_) return;

    // Input the export option and clear the filename if not needed
    ExportFilename_ = "export.txt";
    bool NeedExport = InputExportOption();
    if(QuitOption_) return;
    if(!NeedExport) ExportFilename_.clear();

    // Compute the time step in years and the total number of steps
    TimeStep_ = 1.0/StepsPerYear;
    NumSteps_ = long(Years*StepsPerYear);
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  InputMethodType
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

char Input::InputMethodType()
{
    // Show info
    ut::OutputLine("Welcome to the N-body simulation application");
    ut::OutputLine();
    ut::OutputLine("Available simulation methods:");
    ut::OutputLine("e) Euler discretization");
    ut::OutputLine("r) Runge-Kutta discretization:");
```

```
    // Input the method type
    const std::string prompt = "Method type (e, r, q - quit)";
    const char type = ut::get_char_in_range(prompt, "erq");

    // Quit on q
    if(type == 'q') QuitOption_ = true;
    return type;
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  InputScenarioNumber
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

char Input::InputScenarioNumber()
{
    // Show info
    ut::OutputLine();
    ut::OutputLine("Available scenarios:");
    ut::OutputLine("1) Sun-Earth");
    ut::OutputLine("2) Sun-Venus-Earth-Mars-Jupiter");
    ut::OutputLine("3) Sun-Venus-Earth-Mars perturbed by a passing neutron
                                                        star");
    ut::OutputLine("4) Sun-Venus-Earth-Mars perturbed by a passing neutron
                                                        star (altered)");

    // Input the scenario number
    const std::string prompt = "Scenario (1, 2, 3, 4, q - quit)";
    const char number = ut::get_char_in_range(prompt, "1234q");

    // Quit on q
    if(number == 'q') QuitOption_ = true;
    return number;
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  InputYears
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

double Input::InputYears()
{
    // Show prompt
    const std::string prompt = "How many years to simulate? (q - quit)";
    bool valid = false;
    double value = 0.0;

    // Repeat until valid
    while(!valid)
    {
        // Input a string and convert to double
        std::string input = ut::GetString(prompt);
        value = lv::StringToDouble(input, valid);
        if(value <= 0) valid = false;

        // Quit on q
```

```
        if(input == "q")
        {
            valid = true;
            QuitOption_ = true;
        }
    }

    return value;
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  InputStepsPerYear
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

long Input::InputStepsPerYear()
{
    // Show prompt
    const std::string prompt = "How many time steps per year? (q - quit)";
    bool valid = false;
    long value = 0;

    // Repeat until valid
    while(!valid)
    {
        // Input a string and convert to double
        std::string input = ut::GetString(prompt);
        value = lv::StringToLong(input, valid);
        if(value <= 0) valid = false;

        // Quit on q
        if(input == "q")
        {
            valid = true;
            QuitOption_ = true;
        }
    }

    return value;
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  InputExportOption
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

bool Input::InputExportOption()
{
    // Input the export option
    const std::string prompt = "Export to " + ExportFilename_ + "? (y, n,
                                                                    q - quit)";
    const char flag = ut::get_char_in_range(prompt, "ynq");

    // Quit on q
    if(flag == 'q') QuitOption_ = true;
    return flag == 'y';
}
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Input.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef InputH
#define InputH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include <string>

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class Input
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class Input
{
    public:
        Input();

        char GetMethodType() const {return MethodType_;}
        char GetScenarioNumber() const {return ScenarioNumber_;}

        double GetTimeStep() const {return TimeStep_;}
        long GetNumSteps() const {return NumSteps_;}

        std::string GetExportFilename() const {return ExportFilename_;}
        bool GetQuitOption() const {return QuitOption_;}

    private:
        char InputMethodType();
        char InputScenarioNumber();
        double InputYears();
        long InputStepsPerYear();
        bool InputExportOption();

        char MethodType_;               // Method type
        char ScenarioNumber_;           // Scenario number

        double TimeStep_;               // Simulation time step
        long NumSteps_;                 // Number of simulation steps

        std::string ExportFilename_;    // Filename for exporting
        bool QuitOption_;               // Early exit
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Export.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "Export.h"
#include <sstream>
#include <iomanip>
#include <stdexcept>

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Constructor
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Export::Export(const std::string & filename)
{
    // Open the file for exporting
    stream.open(filename.c_str());

    // Error checking
    if(!stream.is_open())
        throw std::runtime_error("Cannot open file " + filename);
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Registration
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

void Export::RegisterExport(long column, const std::string & label, double value)
{
    // Register the column label
    LabelMap_[column] = label;

    // Register the value converted to text
    std::ostringstream ss;
    ss << value;
    ValueMap_[column] = ss.str();
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Export
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

void Export::DoExport(bool NeedHeader)
{
    // Export column labels at the start
    if(NeedHeader)
    {
        // For each registered entry
        for(MapIt k = LabelMap_.begin(); k != LabelMap_.end(); ++k)
        {
            // Export the registered label with padding
            stream << std::setw(20) << k->second;
        }
```

```
        stream << std::endl;
    }

    // For each registered entry
    for(MapIt k = ValueMap_.begin(); k != ValueMap_.end(); ++k)
    {
        // Export the registered value with padding
        stream << std::setw(20) << k->second;
    }

    stream << std::endl;
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Export.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef ExportH
#define ExportH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include <string>
#include <map>
#include <fstream>

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class Export
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class Export
{
    public:
        explicit Export(const std::string & filename);

        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        //  Registration
        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

        void RegisterExport(long column, const std::string & label, double value);

        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        //  Export
        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

        void DoExport(bool NeedHeader);

    private:
        typedef std::map<long, std::string> Map;    // Column to text mapping
        typedef Map::iterator MapIt;                // Map iterator

        Map LabelMap_;          // Storage for label entries
        Map ValueMap_;          // Storage for value entries
        std::ofstream stream;
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  TableOutput.cpp
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include "TableOutput.h"
#include "utility.h"
#include <iostream>
#include <iomanip>
#include <sstream>

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Registration
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

void TableOutput::RegisterOutput(long row, long column, const std::string & text)
{
    // Add the mapping from (row, column) to text
    const Cell cell = std::make_pair(row, column);
    map_[cell] = text;
}

void TableOutput::RegisterOutput(long row, long column, double value)
{
    // Register the value converted to text
    std::ostringstream ss;
    ss << std::setprecision(4) << value;
    RegisterOutput(row, column, ss.str());
}

void TableOutput::RegisterOutput(long row, long lcolumn, const std::string & label,
                                           long vcolumn, double value)
{
    // Register the label at (row, lcolumn) and value at (row, vcolumn)
    RegisterOutput(row, lcolumn, label);
    RegisterOutput(row, vcolumn, value);
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  Output
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

void TableOutput::DoOutput() const
{
    // Previous row
    long prow = 0;

    // For each registered entry
    for(MapIt k = map_.begin(); k != map_.end(); ++k)
    {
        // Table cell position
        const Cell cell = k->first;
        const long row = cell.first;
        const long column = cell.second;
```

```
        // Go to the next line if different row from the previous
        if(row != prow) std::cout << std::endl;

        // Output with padding, smaller for the first column
        const long width = (column == 1) ? 7 : 12;
        std::cout << std::setw(width) << k->second;

        // Update the previous row
        prow = row;
    }

    std::cout << std::endl;
}

void TableOutput::OutputBanner(const std::string & text) const
{
    ut::OutputLine(text);
}

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  TableOutput.h
//  Assignment 2 - 1422827
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#ifndef OutputH
#define OutputH
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

#include <string>
#include <map>

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  class TableOutput
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

class TableOutput
{
    public:
        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        //  Registration
        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

        void RegisterOutput(long row, long column, const std::string & text);
        void RegisterOutput(long row, long column, double value);
        void RegisterOutput(long row, long lcolumn, const std::string & label,
                                      long vcolumn, double value);
        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        //  Output
        //XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

        void DoOutput() const;
        void OutputBanner(const std::string & text) const;

    private:
        typedef std::pair<long, long> Cell;         // Row/column of the table cell
        typedef std::map<Cell, std::string> Map;    // Row/column to text mapping
        typedef Map::const_iterator MapIt;          // Map iterator

        Map map_;   // Storage for table entries
};

//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
#endif
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
//  End Of File
//XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```