Star Catalogue using Object Oriented Programming Techniques in C++

Qi Nohr Chen - 10499501

PHYS30762 - Object Oriented Programming in C++

May 2023

Abstract

The program uses techniques from object oriented programming as well as advanced C++ features and functionality to create a star catalogue that can be used to browse through various galaxies that a user has read in or written. The advanced C++ features are mainly the use of template functions, name spaces, lambda functions and the use of smart pointers. Advanced functionality includes features such as filtering, sorting, validations, advanced file editing, and searching. The goal of this project was to make it as close to a catalogue as possible for which a user can browse through multiple galaxies and learn more about their constituents.

Contents

1 Introduction

In the field of Astrophysics, the profession is very reliant on computation and data. Nowadays, you won't hear about an Astrophysicist that doesn't know how to manipulate data in different coding languages. The universe consists of many stellar objects, which can all be classified. Their classification is dependent on parameters. A Cepheid variable for example has a period at which it "blinks", a Star itself can be defined by its life cycle stage, and other bodies will also have physical features defining and classifying them.

The star catalogue is a C++ program that reads in data (typed or read as a CSV file) and generates a catalogue that a user can browse to find out more information about specific galaxies and their objects. The objects focused on were neutron stars, planets and stars of type O, B, A, F, G, K, S and M. The goal of this program is to create a sort of catalogue for both the general public and astronomers, for which objects and files can be edited and saved.

The pillars of Object Oriented Programming (now referred to as OOP) are Abstraction, Encapsulation, Inheritance and Polymorphism. Commonly classes can be thought of as "family trees". In this case, there is a base abstract class serving as an interface for the inherited classes called "Celestial Bodies". This branches into two classes, "Galaxy" and "Singular Bodies". The Galaxy-class serves as a container storing singular bodies but also has its own member variables. Singular Bodies branch into 3 types, neutron stars, planets and stars.

The star catalogue program uses several advanced C++ features. These features include template functions, lambda functions, advanced code organization, smart pointers and namespaces. Advanced features allow the code to perform more efficiently and sometimes make the code cleaner. They also improve user experience as the advanced features allow for functionalities within the star catalogue. These advanced functionalities include filtering, sorting, deleting and searching for various objects. Other advanced functionalities include certain validations and other calculations.

2 Code Design and Implementation

2.1 OOP Design and Class Hierarchy

The code has an abstract base class called "CelestialBodies", which provides an interface for the inherited classes. The abstract class branches off into two classes, one is the Galaxy class and the other class is called "SingularBodies". The Galaxy-class acts as a container of SingularBodies with the use of a vector, but it also has its properties such as the mass and name of the galaxy. SingularBodies however branches off into 3 more objects. Planets, Stars and Neutronstars. These are the particular objects that are stored within the galaxy container. The CelestialBodies class contains only virtual functions, the SingularBodies function contains both virtual functions and member functions that the derived classes access. The purpose of the classes inheriting the features of SingularBodies is to keep and output information of differential astronomical bodies.

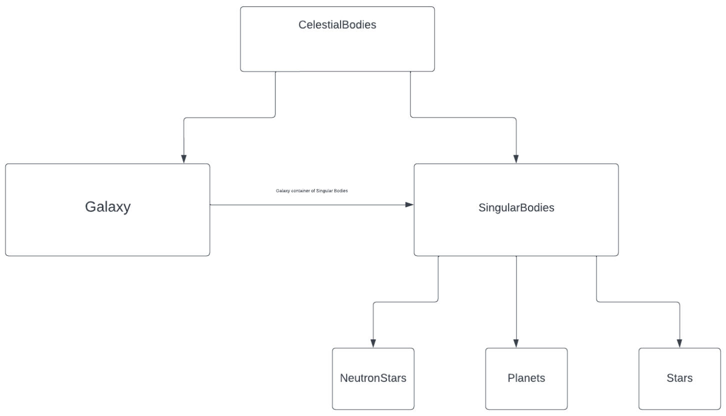


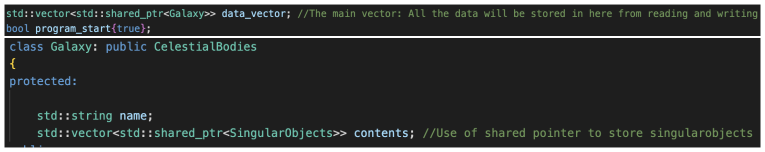
Figure 1: Class hierarchy using a flow diagram.

Other functions within the CelstialBodies family of functions include the access and setting functions. As an example, "get mass" or "set contents" in the Galaxy class, return the mass in the form of a double and set the contents of the SingularBodies vector within the galaxy. Within the SingularBodies a commonly used function is the "get type" function which is used for filtering and returns an std::string of the type of SingularObject (Star, Planet or Neutronstar).

2.2 Advanced C++ Features

2.2.1 Smart Pointers

The use of smart pointers is particularly useful for several reasons. The implementation of smart pointers is fairly easy. Defining a smart pointer does not require much work and they are very practical. A smart pointer deletes itself after use. Deleting new instances is incredibly important in C++. Not deleting new instances can create memory leakages which can lead to loss of available memory and possible computer crashes. The pointer used in this project is a share_pointer. The share_pointer was used together with vectors such as my main data vector and a vector storing singular objects in my galaxy class. The share_pointer allows the ownership of the pointer to be shared, and its automatic deletion is highly convenient to prevent memory leakage.



```
std::vector<std::shared_ptr<Galaxy>> data_vector; //The main vector: All the data will be stored in here from reading and writing
bool program_start{true};
```

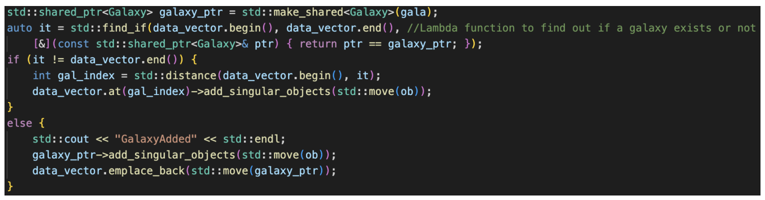
```
class Galaxy: public CelestialBodies
{
protected:

    std::string name;
    std::vector<std::shared_ptr<SingularObjects>> contents; //Use of shared pointer to store singularobjects
```

Figure 2: Snapshot of the main data vector storing galaxies on top and a snapshot of the vector storing singular objects in the galaxy class using shared pointers.

In both of those cases, it is highly convenient to have shared pointers as operations can be performed that are not allowed in, for example, unique pointers, due to ownership restrictions, which are solved with shared pointers.

2.2.2 Lambda Functions

Lambda functions are anonymous short functions. These are commonly used when short operations need to be performed. Another use for it is cleanliness. Having too many lines of code for a simple function is sometimes not efficient. A code has to be readable, and lambda functions allow readability and cleanliness, especially if another person is trying to understand the code that was written. Lambda functions were used to create searching and sorting functions. I have also used lambda functions within other functions to perform simple operations.



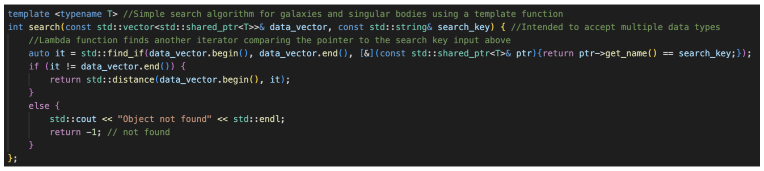
```
std::shared_ptr<Galaxy> galaxy_ptr = std::make_shared<Galaxy>(gala);
auto it = std::find_if(data_vector.begin(), data_vector.end(), //Lambda function to find out if a galaxy exists or not
    [&](const std::shared_ptr<Galaxy>& ptr) { return ptr == galaxy_ptr; });
if (it != data_vector.end()) {
    int gal_index = std::distance(data_vector.begin(), it);
    data_vector.at(gal_index)->add_singular_objects(std::move(ob));
}
else {
    std::cout << "GalaxyAdded" << std::endl;
    galaxy_ptr->add_singular_objects(std::move(ob));
    data_vector.emplace_back(std::move(galaxy_ptr));
}
```

Figure 3: Snapshot of a lambda function that checks whether a galaxy already exists in a vector.

Figure 3 shows the implementation of lambda functions. A simple equivalence operation is used to figure out the existence of a galaxy. The convenience of lambda functions lies in the fact that the function will only be used once in this function. Hence, there is no need to declare a separate function. This keeps coding concise for readers. This isn't the only lambda function that was used. Other lambda functions in the code, such as in my search functions, were used in combination with another advanced C++ feature, which is the template functions.

2.2.3 Template Functions

The template function is, as the name suggests, a template of a function. This means that it allows operations on different data types. In the star catalogue, a common implementation in the star catalogue is the search simple "linear search algorithm" that is intended to accept an input of a vector of share pointers of various data types. In this case, it was intended to accept either Galaxy class objects or Singular body objects. Another use of the template function was the "deleting object function". Once again, a delete function is commonly used in the code to delete either singular bodies or galaxy objects.



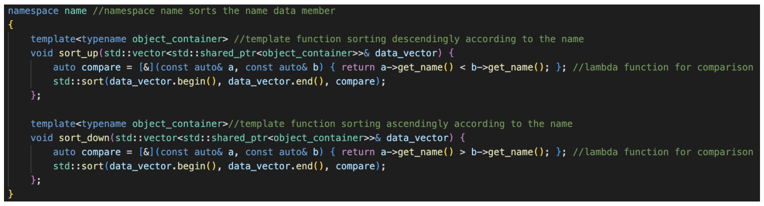
```cpp
template <typename T> //Simple search algorithm for galaxies and singular bodies using a template function
int search(const std::vector<std::shared_ptr<T>>& data_vector, const std::string& search_key) { //Intended to accept multiple data types
    //Lambda function finds another iterator comparing the pointer to the search key input above
    auto it = std::find_if(data_vector.begin(), data_vector.end(), [&](const std::shared_ptr<T>& ptr){return ptr->get_name() == search_key;});
    if (it != data_vector.end()) {
        return std::distance(data_vector.begin(), it);
    }
    else {
        std::cout << "Object not found" << std::endl;
        return -1; // not found
    }
};
```

Figure 4: Snapshot of a template function that checks that searches an object in a vector of shared pointers of any data type.

Figure 4 displays a function that calculates the index of an object in a vector. However, if not found then it will return an index of -1. The choice of using the template function is due to the input of distinct classes that requires the functionality. Using the template, the search and delete functions can pass both, singular bodies or galaxy vectors. However, these aren't the only functions where template functions were used. Template functions were also used together with namespaces, which is another advanced C++ feature, in sorting functions.

2.2.4 Namespaces

Namespaces are advanced C++ features which allow the avoidance of name collisions. Certain functions may have the same name but instead, perform different actions. Instead of changing the name of the function and making slight alterations to it, it is fairly logical to create a namespace. The star catalogue can sort galaxies and singular bodies according to name, mass, radius etc using a function called "sort_up" or "sort_down". The functions have the same name but act differently on those variables due to the defined namespace. It is logical to let those functions have the same name as they perform the same action, and the namespace conveniently tells the person reading the code, which data member the function is sorting. Furthermore, this isn't the only use of namespaces. They also keep the code more organised.



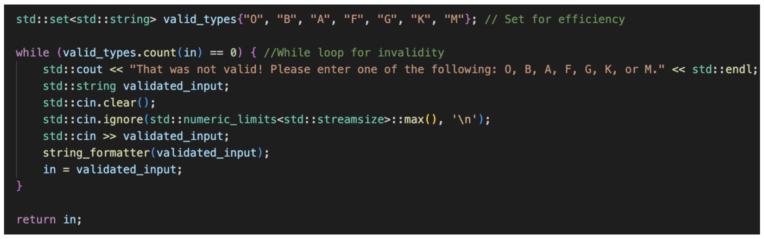
```cpp
namespace name //namespace name sorts the name data member
{
    template<typename object_container> //template function sorting descendingly according to the name
    void sort_up(std::vector<std::shared_ptr<object_container>>& data_vector) {
        auto compare = [&](const auto& a, const auto& b) { return a->get_name() < b->get_name(); }; //lambda function for comparison
        std::sort(data_vector.begin(), data_vector.end(), compare);
    };

    template<typename object_container>//template function sorting ascendingly according to the name
    void sort_down(std::vector<std::shared_ptr<object_container>>& data_vector) {
        auto compare = [&](const auto& a, const auto& b) { return a->get_name() > b->get_name(); }; //lambda function for comparison
        std::sort(data_vector.begin(), data_vector.end(), compare);
    };
}
```

Figure 5: Snapshot of a namespace that uses sort up and sort down functions to avoid name collisions.

The namespace chosen was "name" in this case as it sorts shared pointer vectors by their name. This allows the code to reuse the function name without overriding or overloading the function.

2.2.5 Header files and other Advanced Features

Header files are good coding practice as it keeps the code organised. As iterated many times before, keeping a concise and organized code is very important for readability. A code that is easy to understand is considered good code, and dividing files into header files gives a great overview, especially if the header files were given appropriate names. The header files in the star catalogue contain classes, template functions, namespaces and declarations of function, for which the declared functions are defined in a separate .cpp file.

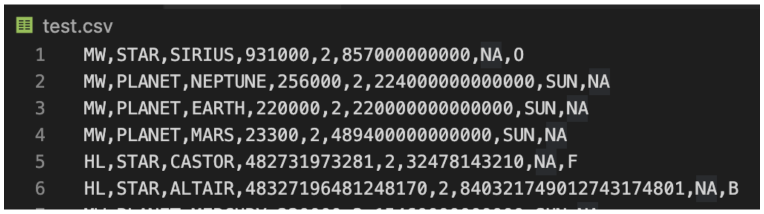


```cpp
std::set<std::string> valid_types{"O", "B", "A", "F", "G", "K", "M"}; // Set for efficiency

while (valid_types.count(in) == 0) { //While loop for invalidity
    std::cout << "That was not valid! Please enter one of the following: O, B, A, F, G, K, or M." << std::endl;
    std::string validated_input;
    std::cin.clear();
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    std::cin >> validated_input;
    string_formatter(validated_input);
    in = validated_input;
}

return in;
```

Figure 6: Snapshot of a the use of sets to check for valid star type input

In Figure 6 Sets were used to increase the time efficiency of the star-type validation function for star type inputs. The building time of a set is $O(n \log n)$ [1]and the validation itself has a time complexity of $O(\log n)$. This is far better than the linear validation using vectors.

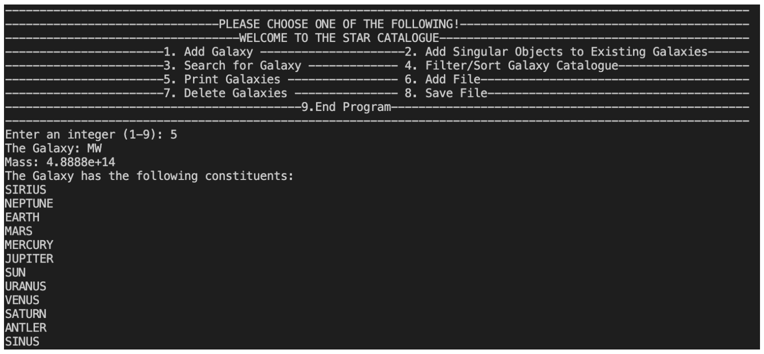
3 Results



```
test.csv
MW,STAR,SIRIUS,931000,2,857000000000,NA,O
MW,PLANET,NEPTUNE,256000,2,224000000000000,SUN,NA
MW,PLANET,EARTH,220000,2,220000000000000,SUN,NA
MW,PLANET,MARS,23300,2,489400000000000,SUN,NA
HL,STAR,CASTOR,482731973281,2,32478143210,NA,F
HL,STAR,ALTAIR,48327196481248170,2,840321749012743174801,NA,B
```

Figure 7: Snapshot of a CSV input (note that this is not the full input).

The user can either create a data set or read in a data set (or multiple) as a CSV file (refer to 7). The data is stored within a main vector of galaxies. The galaxies each have constituents as described in the class hierarchy. The catalogue is designed to have a simple terminal menu and the user can perform various operations and functions on the data vector. The star catalogue allows the user to access different galaxy and their constituents, which would be stars, neutron stars and planets to print out information, similar to an ordinary catalogue of any kind. This can be done through the search function within the code. However, the star catalogue also edits files.



```
-----------------------------------------------------------------------------------------------------------
-------------------------------PLEASE CHOOSE ONE OF THE FOLLOWING!-----------------------------------------
----------------------------------WELCOME TO THE STAR CATALOGUE--------------------------------------------
-----------------------1. Add Galaxy ---------------------2. Add Singular Objects to Existing Galaxies------
-----------------------3. Search for Galaxy ------------- 4. Filter/Sort Galaxy Catalogue-------------------
-----------------------5. Print Galaxies ---------------- 6. Add File---------------------------------------
-----------------------7. Delete Galaxies --------------- 8. Save File--------------------------------------
--------------------------------------------9.End Program--------------------------------------------------
-----------------------------------------------------------------------------------------------------------
Enter an integer (1-9): 5
The Galaxy: MW
Mass: 4.8888e+14
The Galaxy has the following constituents:
SIRIUS
NEPTUNE
EARTH
MARS
MERCURY
JUPITER
SUN
URANUS
VENUS
SATURN
ANTLER
SINUS
```

Figure 8: Snapshot of a terminal output (note that this is not the full output).

The user can choose to create a new file via saving, or sort data vectors. As an example, a user can choose to sort the main vector ascending by their name or mass. The user could also choose to sort the constituents. The functionality was made available through the use of namespaces, lambda functions and template functions. The option to delete objects (singular objects or galaxies) is also available to the user. The extensive functions allow the user to perform a variety of tasks, whether it is to save written star catalogue, edit a star catalogue or browse a star catalogue, the program provides a solid foundation to perform these tasks. However, in any program, improvements can be made. Outputs are created either within the terminal (See Figure 8 for a terminal output) as printed output or as a CSV file.

4 Improvements

An obvious improvement to the program is the user interface (more commonly referred to as UI). Whilst designing the UI, an attempt was made to keep it as readable as possible, however, a UI that doesn't require inputs is considered more user-friendly due to aesthetic reasons, as well as convenience. Besides that, the star catalogue could be greatly improved if the program could provide more graphics. Animations of planets orbiting different stars as well as other movements of objects will be greatly appreciated by any user. The front end and design of the code can be vastly improved, however, given the time, it was considered to be a difficult task. However, with regards to classes, there is no limit to how many classes could have been made as there are several more stars. For example, Cepheid variables, which are significant astronomical objects.

Regarding the functionality, instead of a linear search function, a binary search method could have been used especially given that the star catalogue has a sorting function. A binary search algorithm is more efficient due to its time complexity $O(\log n)$ in big O notation. Another improvement that could be made would be the physical aspect of the code. As of right now, many parameters that the user can enter may seem significantly un-physical. However, the objects are only used to show proof of concepts (specifically concepts of OOP and advanced C++ functionality) rather than the program's quality in solving physical problems.

5 Conclusion

In conclusion, the code can manage data files and edit or sort them. The functionality allows the user to treat the program as a catalogue of stars which they can browse through. The code itself uses a lot of advanced C++ features and uses features to create advanced functionalities such as searching, sorting, deleting etc. Whilst the code does manage to full fill the criteria of a star catalogue, it should be noted that many improvements can be made such as the UI or the physicality of the objects. On a technical aspect, however, the search algorithm can be improved by using the binary search algorithm.

References

[1] Firdous Ahmad Mala and Rouf Ali. The big-o of mathematics and computer science. *Journal of Applied Mathematics and Computation*, 6(1):1–3, 2022.

(Word Count : 2080)