**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**КЫРГЫЗСКОЙ РЕСПУБЛИКИ**

**КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. И.Раззакова**

**ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

Кафедра: **Программное обеспечение компьютерных систем**

Курс «ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ С++»

**СРС**

**Тема: «Справочник астронома»**

Выполнил: студент группы ПИ-3-21

Табылдиева Шоола

Проверил: Мусабаев Э.Б.

**Бишкек 2024**

Оглавление

[Цели работы 2](#_Toc164763938)

[C++ 2](#_Toc164763939)

[Задачи самостоятельной работы 2](#_Toc164763940)

[Решение 2](#_Toc164763941)

[Вывод 5](#_Toc164763942)

# Цели работы

Целью объектно-ориентированного программирования (ООП) является создание программных систем, которые легко понимать, модифицировать и поддерживать. В контексте решения задач ООП целью является разработка эффективного и структурированного программного решения, которое отражает структуру и взаимодействие объектов в реальном мире, упрощает разработку, обеспечивает повторное использование кода и улучшает его читаемость. Основные принципы ООП, такие как инкапсуляция, наследование и полиморфизм, используются для создания моделей, которые отображают реальные объекты и их взаимодействие, что делает программу более понятной и легкой для сопровождения.

# C++

**C++** — это компилируемый, статически типизированный язык программирования, который является расширением языка C. Он поддерживает различные парадигмы программирования, включая процедурное, объектно-ориентированное и обобщенное программирование, что делает его мощным инструментом для разработки разнообразных приложений, от небольших скриптовых программ до крупного системного уровня.

C++ для решения задач ООП:

1. Поддержка объектно-ориентированного программирования: C++ обладает мощными средствами для создания классов и объектов, что позволяет разработчикам эффективно моделировать реальные объекты и их взаимодействие.
2. Инкапсуляция: C++ поддерживает инкапсуляцию, позволяя скрывать детали реализации объектов от пользователей классов. Это способствует разделению ответственности и упрощает разработку сложных систем.
3. Наследование: Одним из основных принципов ООП является наследование, которое позволяет создавать новые классы на основе существующих. C++ обеспечивает механизмы наследования, позволяющие создавать иерархии классов для повторного использования кода и создания иерархических структур.
4. Полиморфизм: C++ поддерживает полиморфизм, что означает, что одинаково названные методы могут выполнять различные действия в зависимости от типа объекта, к которому они применяются. Это способствует созданию гибких и расширяемых программ.
5. Эффективность: C++ является компилируемым языком, что означает, что код компилируется в машинный код, что обеспечивает высокую производительность и эффективность исполнения программ.

# Задачи самостоятельной работы

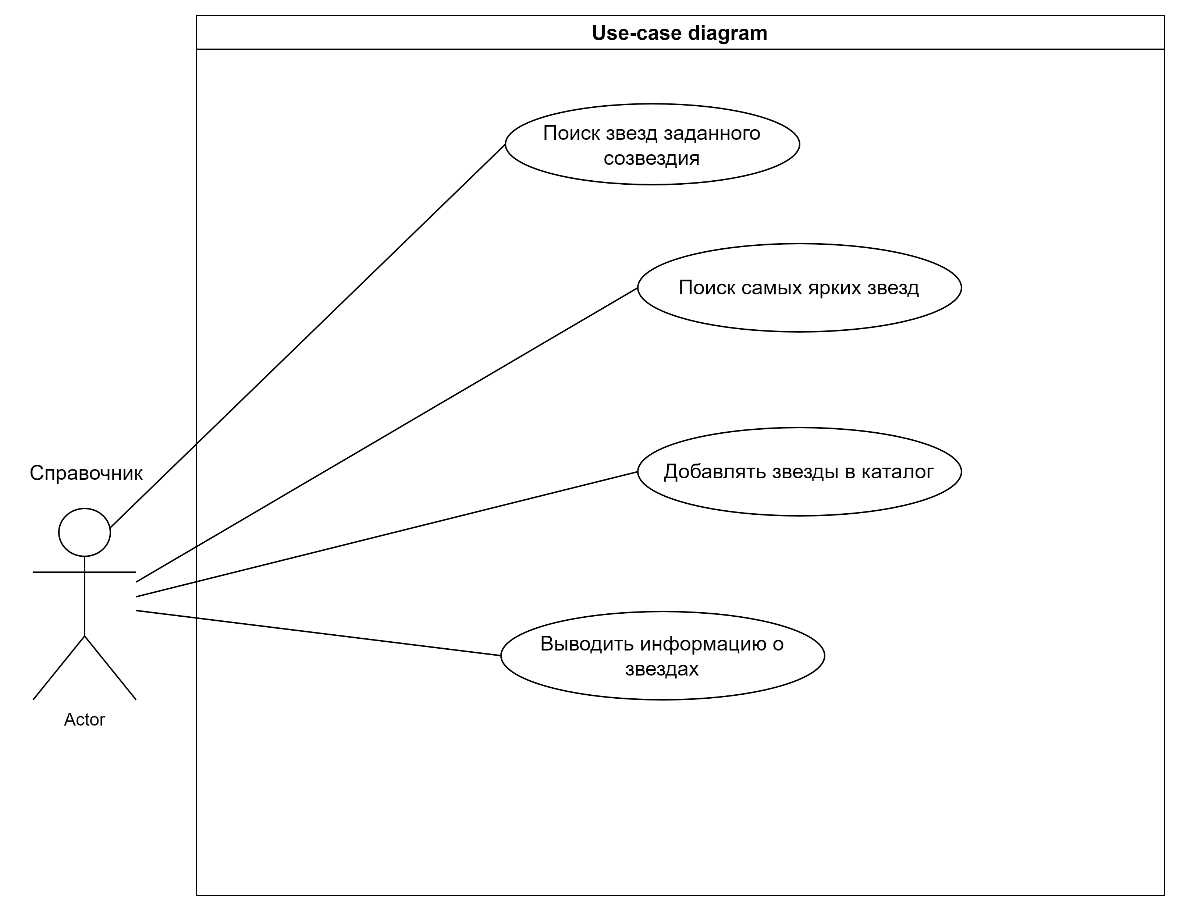
Реализовать программу с использованием принципов ООП:

Справочник астронома. Для каждой из зарегистрированных звезд известны: название, созвездие, видимая звездная величина, расстояние, координаты на небосклоне. Поиск звезд заданного созвездия и самых ярких звезд.

# Решение

Наша программа является консольным приложением. Написан был на языке C++.

1. **Use-Case диаграмма**



1. **Листинг кода:**

#include <iostream>

#include <vector>

#include <string>

#include <Windows.h>

#include <algorithm>

using namespace std;

// Класс для звезды

class Star {

private:

string name; //названия звезды

string constellation; //названия созвездия, в котором находится звезда

double magnitude; //звездная величина

double distance; //расстояние до звезды

pair<double, double> coordinates; // Координаты на небесной сфере

public:

Star(string \_name, string \_constellation, double \_magnitude, double \_distance, pair<double, double> \_coordinates) :

name(\_name), constellation(\_constellation), magnitude(\_magnitude), distance(\_distance), coordinates(\_coordinates) {}

string getName() const { return name; }

string getConstellation() const { return constellation; }

double getMagnitude() const { return magnitude; }

double getDistance() const { return distance; }

pair<double, double> getCoordinates() const { return coordinates; }

};

// Класс для справочника звезд

class StarCatalog {

private:

vector<Star> stars;

public:

// Метод для добавления звезды в справочник

void addStar(const Star& star) {

stars.push\_back(star);

}

// Метод для поиска звезды по созвездию

vector<Star> findStarsByConstellation(string constellation) const {

vector<Star> result;

for (const auto& star : stars) {

if (star.getConstellation() == constellation) {

result.push\_back(star);

}

}

return result;

}

// Метод для поиска самых ярких звезд в справочнике

vector<Star> findBrightestStars(int count) const {

vector<Star> result;

vector<Star> sortedStars = stars;

sort(sortedStars.begin(), sortedStars.end(), [](const Star& a, const Star& b) {

return a.getMagnitude() < b.getMagnitude();

});

for (int i = 0; i < count && i < sortedStars.size(); ++i) {

result.push\_back(sortedStars[i]);

}

return result;

}

// Метод для вывода информации о звездах

void printStars(const vector<Star>& stars) const {

for (const auto& star : stars) {

cout << "Name: " << star.getName() << ", Constellation: " << star.getConstellation() << ", Magnitude: "

<< star.getMagnitude() << ", Distance: " << star.getDistance() << ", Coordinates: ("

<< star.getCoordinates().first << ", " << star.getCoordinates().second << ")" << endl;

}

}

};

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

StarCatalog catalog;

// Добавляем звезды

catalog.addStar(Star("Sirius", "Canis Major", -1.46, 8.6, make\_pair(101.28, -16.72)));

catalog.addStar(Star("Vega", "Lyra", 0.03, 25.0, make\_pair(279.23, 38.78)));

catalog.addStar(Star("Altair", "Aquila", 0.77, 16.7, make\_pair(297.7, 8.87)));

// Поиск звезд по созвездию

cout << "Stars in Canis Major:" << endl;

catalog.printStars(catalog.findStarsByConstellation("Canis Major"));

cout << endl;

// Поиск самых ярких звезд

cout << "Brightest stars:" << endl;

catalog.printStars(catalog.findBrightestStars(3));

return 0;

}

1. **Описание полей и методов:**

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Класс** | **Имя поля/метода** | **Описание** |
| 1 | Star | string name | Поле для названия |
| 2 | Star | string constellation | Поле для созвездия |
| 3 | Star | double magnitude | Поле для видимой звездной величины |
| 4 | Star | double distance | Поле для расстояния |
| 5 | Star | pair<double, double> coordinates | Поле для координатов на небосклоне |
| 6 | StarCatalog | vector<Star> stars | Поле для хранения звезд в справочнике |
| 7 | StarCatalog | void addStar(const Star& star) | Метод для добавления звезды в справочник |
| 8 | StarCatalog | vector<Star> findStarsByConstellation(string constellation) const | Метод для поиска звезды по созвездию |
|  | StarCatalog | vector<Star> findBrightestStars(int count) const | Метод для поиска самых ярких звезд в справочнике |
| 9 | StarCatalog | void printStars(const vector<Star>& stars) const | Метод для вывода информации о звездах |

1. **Результат выполнения программы (рис. 4.1):**

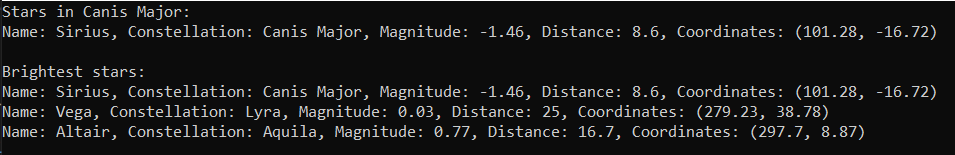


Рисунок 4.1 Результат выполнения программы

# Вывод

В данной программе мы использовали следующие принципы объектно-ориентированного программирования (ООП):

* **Использование классов**: Мы создали классы **Star** и **StarCatalog**, чтобы абстрагировать данные о звездах и справочнике звезд соответственно.
* **Инкапсуляция**: Данные о каждой звезде инкапсулированы в классе **Star**, а методы для работы с справочником звезд инкапсулированы в классе **StarCatalog**.

В результате создания данной программы мы приобрели опыт в применении принципов ООП для решения конкретных задач, а также углубились в работу с языком C++, который обеспечивает высокую производительность и мощные инструменты для разработки программного обеспечения.