НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ
«МИФИ»

Институт ЛаПлаз

Кафедра №31 «Прикладная математика»

ЛАБОРАТОРНАЯ РАБОТА №5

по дисциплине

«Объектно-ориентированное программирование»

Выбор варианта

Номер варианта определяется по формуле: `n = (k mod 2) + 1`, где `k` — номер студента в списке группы.

Вариант 1. Игра «Камень, Ножницы, Бумага»

В данной лабораторной работе требуется реализовать классическую игру «Камень-Ножницы-Бумага», используя принципы объектно-ориентированного программирования, в частности механизм динамического полиморфизма.

Основой архитектуры программы должен стать абстрактный базовый класс `Item`, который будет представлять обобщённый игровой элемент. Данный класс должен содержать `protected` поле для хранения имени объекта (например, строку `std::string name`) и методы доступа к нему. От базового класса `Item` необходимо унаследовать три конкретных класса: `Rock` (камень), `Paper` (бумага) и `Scissors` (ножницы). Каждый из этих классов должен реализовывать специфичное для него поведение в соответствии с правилами игры: камень побеждает ножницы, ножницы побеждают бумагу, а бумага побеждает камень. Для этого требуется реализовать перегрузки операторов сравнения `<`, `>`, `==` и `!=` для объектов типа `Item`. Эти операторы должны позволять сравнивать различные игровые элементы между собой, определяя победителя в раунде. Для реализации данной функциональности потребуется использовать механизм динамического полиморфизма. Для идентификации типов объектов во время выполнения программы, чтобы не привязываться к имени, рекомендуется использовать оператор `typeid`[1], который позволяет получить информацию о фактическом типе объекта.

Программа должна реализовывать игровой процесс между человеком и ботом. Пользователь выбирает один из трёх вариантов через консольный интерфейс, после чего бот делает свой выбор. Для простейшей реализации бот может использовать генератор случайных чисел для выбора своего хода. После каждого раунда программа должна определять победителя, используя перегруженные операторы сравнения, и выводить результат на экран. Рекомендуется вести подсчёт общего счёта игры и предоставлять возможность играть несколько раундов подряд.

Дополнительное задание

Исследования поведения людей в игре «Камень-Ножницы-Бумага» выявили интересные зависимости. Статистически установлено, что большинство игроков начинают игру с выбора камня. После выигрыша игроки склонны повторять успешный выбор, а после проигрыша менять стратегию. При смене выбора наблюдается тенденция движения по циклу: *Камень* → *Бумага* → *Ножницы* → *Камень*. Также замечено, что люди редко повторяют один и тот же выбор более трёх-четырёх раз подряд, даже при серии побед.

В качестве дополнительного задания предлагается реализовать бота, который будет имитировать человеческое поведение. Посмотрите в интернете, какие существуют стратегии для более частой победы в данной игре и проверьте их на данном боте.

[1]Оператор `typeid` возвращает ссылку на объект типа `std::type_info`, содержащий информацию о типе выражения. Подробнее: `https://en.cppreference.com/w/cpp/language/typeid`

Вариант 2. Игра «Алхимия»

В этом варианте предлагается реализовать упрощенную версию игры «Алхимия». Суть игры заключается в комбинировании базовых элементов для получения новых веществ.

Необходимо разработать иерархию классов с абстрактным базовым классом `Element`. Этот класс должен содержать `protected` поле для хранения имени элемента, а также соответствующие методы доступа к нему. От базового класса наследуются четыре основных природных элемента: `Fire` (Огонь), `Water` (Вода), `Earth` (Земля), `Air` (Воздух).

Основная задача состоит в реализации механизма взаимодействия двух элементов через перегрузку оператора сложения `operator+`. Оператор должен принимать два указателя на объекты типа `Element` и возвращать указатель на новый объект, если реакция возможна, или нулевой указатель (`nullptr`), если элементы не взаимодействуют.

Для идентификации типов объектов во время выполнения программы, чтобы не привязываться к имени, рекомендуется использовать оператор `typeid`[2], который позволяет получить информацию о фактическом типе объекта.

Таблица рецептов (обязательная, но можете расширить):

- `Fire` + `Water` → `Steam` (Пар)
- `Earth` + `Water` → `Mud` (Грязь)
- `Fire` + `Earth` → `Lava` (Лава)
- `Air` + `Fire` → `Energy` (Энергия)

Примечание: порядок сложения не должен иметь значения (т.е. Огонь + Вода должно давать тот же результат, что и Вода + Огонь).

Игровой процесс должен выглядеть так: у пользователя есть набор имеющихся элементов. Изначально там только базовые. Пользователь выбирает индексы двух элементов из списка. Если комбинация удачная, новый элемент добавляется в конец списка, и игра продолжается. Исходные элементы-ингредиенты не расходуются и могут быть использованы повторно.

Дополнительное задание

Измените логику работы программы так, чтобы элементы тратились при взаимодействии. Реализуйте возможность разделения сложных элементов обратно на составляющие. Для этого перегрузите оператор вычитания `operator-`. При применении этой операции к сложному элементу (например, `Steam`), он должен удаляться из инвентаря, а его составляющие (`Fire` и `Water`) добавятся в конец списка. Если применить оператор к базовому элементу, ничего происходить не должно.

[2]Оператор `typeid` возвращает ссылку на объект типа `std::type_info`, содержащий информацию о типе выражения. Подробнее: `https://en.cppreference.com/w/cpp/language/typeid`