1.

Реализовать класс квадратичной функции SquareFunction. Полями класса являются коэффициенты квадратичной функции a, b, c типа double. По умолчанию коэффициенты должны инициализироваться следующими значениями: a = 1.0, b = -2.0, c = 1.0. Реализовать в классе методы: void setCoefficients (double a, double b, double c) - задает коэффициенты квадратичной функции; double calculate(double x) - рассчитывает значение квадратичной функции в точке x; void printFunction() - выводит функцию в терминал в виде “(a) * x^2 + (b) * x + (c)”, где на место a, b, c подставляются значения коэффициентов, которые хранятся в классе.

Реализация класса должна быть разделена на заголовочный файл и файл с реализацией. В функции main необходимо считать из терминала коэффициенты функции, создать объект функции и задать считанные коэффициенты. После этого необходимо считать два целых числа x1, x2 (x1 < x2), вывести функцию в терминал методом printFunction, затем вывести значения функции с шагом 1 на интервале [x1, x2].

2.

Реализовать класс стека для целых чисел MyStack. В качестве полей класс хранит максимальный размер стека (capacity), текущее количество значений в стеке (size) и динамический массив тип int* (data), где будут хранится элементы стека.

Реализовать конструктор, который принимает целое число - максимальный размер стека. В конструкторе должна выделяться память под стек, текущее количество элементов устанавливается равным 0. В конструкторе добавить вывод сообщения “Create stack with capacity = n”, где вместо n подставляется максимальный размер стека. Реализовать деструктор, который должен очистить память и вывести сообщение “Destroy stack”.

Реализовать конструктор копирования и оператор присваивания с копированием, который копирует размеры и данные стека так, чтобы изменения одного стека не влияло на другой. Добавить в них вывод сообщений “Copy constructor” и “Copy =”.

Реализовать конструктор перемещения и оператор присваивания с перемещением, используя std::swap (в конструкторе не забывайте выделить память). Добавить в них вывод сообщений “Move constructor” и “Move =”.

Реализовать метод добавления в стек void push(int x), который добавляет элемент в стек, записывая его в массив и увеличивая значение количества элементов. Если добавить нельзя, то выбрасывается исключение типа std::runtime_error.

Реализовать метод получения элемента из стека double pop(), который достает элемент сверху стека и уменьшает значение количества элементов. Если стек пустой, , то выбрасывается исключение типа std::runtime_error.

Реализовать метод void state(), который выводит сообщение “Stack with n elements: “, где n текущее количество элементов, и после двоеточия выводятся значения текущих элементов.

В функции main, инициализировать стек размером считанным из терминала, затем проверить операции копирования и перемещения, используя методы push, pop, state, что изменения одного стека не отображаются на том объекте, откуда было копирование/перемещение. Для явного вызова перемещения используйте std::move.

3.

Реализуйте класс SumProdPrinter. Класс имеет два целочисленных поля x и y. Реализуйте конструктор, который принимает два целых числа, и записывает значения в поля x и y.

Реализуйте метод int sum(), который возвращает сумму x и y.

Реализуйте метод int prod(), который возвращает произведение x и y.

Реализуйте метод void printResults(), который выводит в терминал через пробел результат суммы и произведения для чисел x и y.

Реализуйте класс SubSumProdPrinter, который наследуется от SumProdPrinter. В этот класс по аналогии с sum и prod добавьте метод, возвращающий разницу элементов.

Переопределите метод printResults так, чтобы он выводил строку “Result is ”, и далее выводился результат разницы, суммы и произведения.

Попытайтесь сделать так, чтобы свести к минимуму дублирование кода.

В функции main, сделайте бесконечный цикл, в котором считывается один символ и два числа, если был введен символ ‘a’, то нужно вывести сумму и произведение для двух чисел, если символ ‘b’, то нужно вывести “Result is “ И разницу, сумму и произведение для двух чисел, если символ ‘f’, то завершить цикл. в остальных случаях перейти на следующую итерацию цикла.

4.

Создайте абстрактный класс Vehicle, который хранит информацию о бренде и модели транспорта в виде строки, а также год выпуска транспорта. Реализуйте конструктор для заполнения полей. Реализуйте метод displayInfo, который выводит информацию о транспорте в виде “бренд: модель (года выпуска)”. Добавьте два чисто виртуальных метода, double calculateCost(double distance), который должен рассчитать стоимость поездки в зависимости от дистанции в км, и string getType(), который должен вернуть тип транспорта в виде строки.

Сделайте наследников класса Vehicle: Car, Bicycle, ElectricScooter.

Класс Car для типа возвращает “Car”. Имеет дополнительное поле double fuelConsumption, которое хранит потребляеимое количество топлива в литрах на 100 км, и double fuelPrice, которое хранит стоимость топлива. Стоимость поездки рассчитывается как “расстояние/100 * потребление топлива * стоимость топлива”.

Класс Bicycle для типа возвращает “Bicycle”. Стоимость поездки фиксирована и равна 200.

Класс ElectricScooter для типа возвращает “ElectricScooter ”. Имеет дополнительно поле стоимости аккумулятор double batteryCost. Стоимость рассчитывается как стоимость аккумулятора + дистанция * 20.

Создайте класс RentInfo. Класс в качестве поля хранит указатель на Vehicle, который должен передаваться в конструкторе. И сделайте метод showInfo(double distance). Данный метод должен вывести информацию о транспорте, которое хранится в указателе, и стоимость поездки на переданную в качестве аргумента дистанцию.

В функции main создайте массив Vehicle* vehicles[3]. Заполните его по 1-му объекту конкретного транспорта с разными характеристиками. Потом считайте дистанцию планируемой поездки, и используя класс RentInfo, выведите информацию о каждом транспорте в массиве vehicles.

5.

Разработка модульной системы обработки платежей.

Создайте интерфейс платежного шлюза IPaymentGateway. Создайте чисто виртуальные методы bool processPayment(double amount, const std::string& cardNumber) - провести платеж, и std::string getGatewayName() - название способа платежа.

Создайте интерфейс логгера ILogger. Создайте чисто виртуальные методы void logInfo(const std::string& message) - вывод информации, void

logError(const std::string& message) - вывод ошибки, void logTransaction(int transactionId, double amount, const std::string& status) - вывод информации о транзакции.

Создайте интерфейс валидатора IValidator. Создайте виртуальные методы bool validateCard(const std::string& cardNumber) - проверка номеры карты, bool validateAmount(double amount) - проверка баланса счета, std::string getLastError() - функция возвращающая последнюю ошибку.

Реализуйте конкретные платежные шлюзы StripeGateway и PayPalGateway. Метод getGatewayName должен вернуть “Stripe” и “PayPalGateway” соответственно. В методах processPayment, должно выводится сообщение о размере платежа (amount), каким способом и последних 4 цифрах номера карты (cardNumber). StripeGateway возвращает true, если размер платежа больше нуля, и длина номера карты 16 (cardNumber). PayPalGateway возвращает true, если размер платежа больше 0 и меньше 10000.

Реализуйте конкретные логгеры FileLogger и DatabaseLogger. Метод logInfo выводит сообщение в std::cout с припиской “[INFO]” для FileLogger и “[DB-INFO]” для DatabaseLogger. Метод logError аналогично выводит сообщение, но с припиской “[ERROR]” и “[DB-ERROR]”. Метод logTransaction для FileLogger выводит сообщение вида “[TRANSACTION] ID: <transactionId> | Amount: <amount> | Status: <status>”, а для DatabaseLogger сообщение вида “[DB-TRANSACTION] Saved: <transactionId> - Amount: <amount> -Status: <status>”.

Реализуйте конкретные валидаторы BasicValidator и AdvancedValidator, добавив в каждый класс поле строкового типа std::string lastError.

Для BasicValidator метод validateCard должен возвращать false если номер карты не равен 16 символам (в поле lastError записывается строка “Card number must be 16 digits”), или если состоит не только из цифры (в поле lastError записывается строка “Card number must contain only digits”), метод std::isdigit проверяет символ, является ли он цифрой, в остальных случаях возвращает true.

Для AdvancedValidator метод validateCard должен возвращать false если номер состоит не только из цифры (в поле lastError записывается

строка “Card number must contain only digits”), или начинается не с цифры 4 (в поле lastError записывается строка “Invalid card number”), в остальных случаях возвращается true.

Метод validateAmount для обеих реализаций возвращает false, если размер платежа меньше 0 (в поле lastError записывается строка “Amount must be positive”) или больше 50000 (в поле lastError записывается строка “Amount exceeds maximum limit”).

Метод getLastError в обеих реализациях возвращает lastError.

Создайте основной класс проведения платежей PaymentProcessor. В качестве полей он содержит интерфейсы способа платежа IPaymentGateway* paymentGateway, логгера ILogger* logger, валидатора IValidator* validator, а также номер последней транзакции int lastTransaction.

В конструкторе PaymentProcessor принимает указатели IPaymentGateway*, ILogger*, IValidator*, и записывает в соответствующие поля. lastTransaction инициализируется нулем.

Реализуейте метод bool processPayment(double amount, const std::string& cardNumber), который выполняет процесс платежа. Этот метод принимает размер платежа amount и номер карты cardNumber. Состоит из следующих шагов:

1. Логирование через logInfo сообщения "Starting payment processing...".
2. Через валидатор проверяет размер платежа. Если там возникает ошибка, то логирует через logError сообщение “Amount validation failed:” и текст последней ошибки валидатора, а затем завершает метод вернув false.
3. Через валидатор проверяет номер карты. Если там возникает ошибка, то логирует через logError сообщение “Card validation failed:” и текст последней ошибки валидатора, а затем завершает метод вернув false.
4. Далее проводит платеж через шлюз платежа, метод processPayment (запоминает результат платежа в bool переменную).
5. Создается новый номер транзакции (lastTransaction + 1, этот результат также запоминает в lastTransaction).
6. Через логгер и метод logTransaction сообщается информация. В качестве статуса передается строка "SUCCESS", если транзакция прошла успешно, и "FAILED" в ином случае.

7. Из метода возвращается результат платежа.

Реализуйте сеттеры в классе PaymentProcessor для платежного шлюза, логгера и валидатора, чтобы у существующего объекта можно было менять компоненты.

В функции main, создайте объект PaymentProcessor, настроив его конкретными реализациями, проведите несколько транзакций. Замените отдельные компоненты, и повторите транзакции.

Дополнительно:

Создайте класс фабрики компонентов PaymentSystemFactory. У фабрики следующие методы:

- IPaymentGateway* createGateway(const std::string& type). Возвращает объект StripeGateway, если переданная строка “stipe”, и объект PayPalGateway, если переданная строка “paypal”. В остальных случаях кидает исключение.
- ILogger* createLogger(const std::string& type). Возвращает объект FileLogger, если передана строка “file”, и объект DatabaseLogger, если передана строка “database”. В остальных случаях кидает исключение.
- IValidator* createValidator(const std::string& type). Возвращает объект BasicValidator, если передана строка “basic”, и объект AdvancedValidator, если передана строка “advanced”. В остальных случая кидает исключение.

Измените настройку объекта PaymentProcessor, через только что созданную фабрику, а не прямое создание компонентов.