**Эти задачи охватывают различные аспекты программирования на Solidity, начиная от базовых тем до продвинутых структур данных и алгоритмов. Они помогут разработчикам повысить свои навыки и лучше понять возможности Solidity.**

**Уровень 1: Взаимодействие с другими контрактами и расширенные функции**

**Задача 1: Контракт для взаимодействия с другим контрактом** Создай контракт, который взаимодействует с другим контрактом для получения данных.

### Контракт, который хранит данные (DataProvider.sol)

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract DataProvider {

// Хранит данные

uint256 private data;

// Конструктор для установки начального значения данных

constructor(uint256 \_initialData) {

data = \_initialData;

}

// Функция для установки нового значения данных

function setData(uint256 \_data) external {

data = \_data;

}

// Функция для получения текущего значения данных

function getData() external view returns (uint256) {

return data;

}

}

### Контракт, который взаимодействует с DataProvider (DataConsumer.sol)

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Интерфейс для взаимодействия с контрактом DataProvider

interface IDataProvider {

function getData() external view returns (uint256);

}

contract DataConsumer {

// Переменная для хранения ссылки на контракт DataProvider

IDataProvider public dataProvider;

// Конструктор для установки адреса контракта DataProvider

constructor(address \_dataProviderAddress) {

dataProvider = IDataProvider(\_dataProviderAddress);

}

// Функция для получения данных из контракта DataProvider

function fetchData() external view returns (uint256) {

// Вызов функции getData() контракта DataProvider и возврат результата

return dataProvider.getData();

}

// Функция для изменения адреса контракта DataProvider

function setDataProvider(address \_dataProviderAddress) external {

dataProvider = IDataProvider(\_dataProviderAddress);

}

}

Вариант 2

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract DataProvider {

uint256 private data;

constructor(uint256 \_initialData) {

data = \_initialData;

}

function setData(uint256 \_data) external {

data = \_data;

}

function getData() external view returns (uint256) {

return data;

}

}

contract DataConsumer {

// Взаимодействие с DataProvider через call

function fetchData(address dataProviderAddress) external view returns (uint256) {

// Определение сигнатуры функции getData() для call

(bool success, bytes memory data) = dataProviderAddress.staticcall(

abi.encodeWithSignature("getData()")

);

require(success, "Call failed");

// Декодирование возвращаемого значения

return abi.decode(data, (uint256));

}

// Пример вызова функции setData через call

function setData(address dataProviderAddress, uint256 \_newData) external {

// Определение сигнатуры функции setData(uint256) для call

(bool success, ) = dataProviderAddress.call(

abi.encodeWithSignature("setData(uint256)", \_newData)

);

require(success, "Call failed");

}

}

Вариант 3

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract DataProvider {

uint256 public data;

function setData(uint256 \_data) external {

data = \_data;

}

function getData() external view returns (uint256) {

return data;

}

}

contract DataConsumer {

uint256 public data;

// Используем delegatecall для вызова setData в контексте DataConsumer

function setData(address dataProviderAddress, uint256 \_newData) external {

// Определение сигнатуры функции setData(uint256) для delegatecall

(bool success, ) = dataProviderAddress.delegatecall(

abi.encodeWithSignature("setData(uint256)", \_newData)

);

require(success, "Delegatecall failed");

}

// Проверяем, что значение данных изменилось

function getData() external view returns (uint256) {

return data;

}

}

**Задача 2: Контракт для наследования** Напиши базовый контракт и контракт, который его наследует и расширяет функциональность.

Вы правы, в предыдущем ответе я случайно удалил русскоязычные комментарии.

## Base Contract

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Базовый контракт

contract BaseContract {

// Значение

uint256 public value;

// Конструктор, инициализирующий значение

constructor(uint256 \_initialValue) {

value = \_initialValue;

}

// Функция для установки значения

function setValue(uint256 \_newValue) public {

value = \_newValue;

}

// Функция для получения значения

function getValue() public view returns (uint256) {

return value;

}

}

content\_copy Use code [with caution](https://support.google.com/legal/answer/13505487).Solidity

## Inherited Contract

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "./BaseContract.sol";

// Наследуемый контракт

contract InheritedContract is BaseContract {

// Текст

string public text;

// Конструктор, вызывающий конструктор базового контракта и инициализирующий текст

constructor(uint256 \_initialValue, string memory \_initialText)

BaseContract(\_initialValue)

{

text = \_initialText;

}

// Функция для установки текста

function setText(string memory \_newText) public {

text = \_newText;

}

// Функция для получения текста

function getText() public view returns (string memory) {

return text;

}

// Переопределение функции setValue из базового контракта

function setValue(uint256 \_newValue) public override {

// Дополнительная логика перед установкой значения

require(\_newValue > 10, "Value must be greater than 10");

super.setValue(\_newValue);

}

}

content\_copy Use code [with caution](https://support.google.com/legal/answer/13505487).Solidity

**Описание:**

* **BaseContract:**
  + Хранит переменную value.
  + Имеет конструктор, который инициализирует value.
  + Имеет функции setValue and getValue для работы с переменной value.
* **InheritedContract:**
  + Наследует BaseContract, получая доступ к его переменным и функциям.
  + Добавляет переменную text.
  + Имеет конструктор, который вызывает конструктор BaseContract и инициализирует text.
  + Добавляет функции setText and getText для работы с переменной text.
  + Переопределяет функцию setValue из BaseContract, добавляя дополнительную логику валидации перед ее вызовом с помощью super.setValue(\_newValue).

**В результате, InheritedContract:**

* Имеет доступ к переменной value и функциям setValue (с новой логикой) и getValue из BaseContract.
* Имеет собственную переменную text и функции setText и getText.

Этот пример демонстрирует базовые принципы наследования в Solidity, позволяющие создавать новые контракты на основе существующих, расширяя и модифицируя их функциональность.

**Задача 3: Контракт с функцией обратного вызова (fallback)** Создай контракт, который использует функцию fallback для обработки полученных эфиров.

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract FallbackExample {

// Событие для логирования получения эфира

event ReceivedEther(address sender, uint256 amount);

// Функция обратного вызова, вызывается при получении эфира

fallback() external payable {

// Логирование получения эфира

emit ReceivedEther(msg.sender, msg.value);

}

// Функция receive, вызывается при получении эфира без данных

receive() external payable {

// Логирование получения эфира

emit ReceivedEther(msg.sender, msg.value);

}

}

**Описание:**

* **fallback() external payable**:
  + Эта функция срабатывает, когда контракт получает эфир (Ether), но вызов не соответствует ни одной из других функций контракта, либо данные в вызове не могут быть декодированы.
  + Модификатор payable необходим, чтобы контракт мог принимать эфир.
  + Внутри функции логируется событие ReceivedEther с адресом отправителя (msg.sender) и количеством полученного эфира (msg.value).
* **receive() external payable**:
  + Эта функция срабатывает, когда контракт получает эфир без каких-либо дополнительных данных в вызове.
  + Также требует модификатор payable.
  + В данном случае логика функции идентична fallback(), но в реальном контракте вы можете обрабатывать по-разному получение эфира с данными и без.

**Пример использования:**

1. **Отправка эфира на адрес контракта без указания функции и данных:** вызовется функция receive().
2. **Отправка эфира на адрес контракта с вызовом несуществующей функции или с некорректными данными:** вызовется функция fallback().

**Важно:**

* Контракт может иметь только одну функцию fallback() и одну функцию receive().
* Будьте внимательны при работе с функциями fallback() и receive(), так как они могут быть использованы для атак, если не реализованы правильно.

**Задача 4: Контракт с передачей данных в функцию обратного вызова (receive)** Создай контракт, который использует функцию receive для обработки полученных эфиров.

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract ReceiveEther {

// Событие для логирования получения эфира

event LogReceivedEther(address sender, uint256 amount);

// Публичная переменная для хранения баланса контракта

uint256 public contractBalance;

// Функция receive для обработки полученных эфиров

receive() external payable {

contractBalance += msg.value; // Увеличиваем баланс контракта

emit LogReceivedEther(msg.sender, msg.value); // Логируем событие

}

// Функция для получения баланса контракта

function getBalance() public view returns (uint256) {

return contractBalance;

}

}

**Описание:**

1. **receive() функция:**
   * Объявлена с модификатором external payable, что позволяет ей принимать эфир.
   * Увеличивает значение переменной contractBalance на количество полученного эфира (msg.value).
   * Вызывает событие LogReceivedEther, чтобы записать в лог информацию о транзакции.
2. **contractBalance переменная:**
   * Хранит текущий баланс контракта в wei.
   * Объявлена как public, что позволяет читать ее значение извне контракта.
3. **getBalance() функция:**
   * Возвращает текущий баланс контракта.

**Как это работает:**

Когда кто-то отправляет эфир на адрес этого контракта без указания функции или данных, автоматически вызывается функция receive().

**Пример использования:**

1. Разверните этот контракт в сети Ethereum.
2. Отправьте некоторое количество эфира на адрес контракта.
3. Проверьте баланс контракта с помощью функции getBalance(). Баланс должен увеличиться на отправленную вами сумму.

**Важно:**

* Контракт может иметь только одну функцию receive().
* Функция receive() не может иметь параметров и не может возвращать значения.
* Убедитесь, что ваша логика обработки эфира в функции receive() безопасна и не подвержена уязвимостям.

**Уровень 2: Расширенные маппинги и структуры данных**

**Задача 5: Контракт с вложенными маппингами** Создай контракт, который будет использовать вложенные маппинги для хранения данных.

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

contract NestedMappings {

// Вложенный маппинг

// Ключ 1 уровня - адрес (address)

// Ключ 2 уровня - строка (string)

// Значение - число (uint256)

mapping(address => mapping(string => uint256)) public data;

// Функция для установки значения в маппинге

function setValue(address \_user, string memory \_key, uint256 \_value) public {

data[\_user][\_key] = \_value;

}

// Функция для получения значения из маппинга

function getValue(address \_user, string memory \_key) public view returns (uint256) {

return data[\_user][\_key];

}

}

Описание:

data: Вложенный маппинг.

Первый уровень использует адреса (address) в качестве ключей.

Второй уровень использует строки (string) в качестве ключей.

Значение, хранящееся по комбинации ключей - число (uint256).

setValue(): Позволяет записывать данные в маппинг.

Принимает адрес пользователя (\_user), ключ строки (\_key) и значение (\_value).

Устанавливает значение \_value по указанным ключам в маппинге data.

getValue(): Позволяет читать данные из маппинга.

Принимает адрес пользователя (\_user) и ключ строки (\_key).

Возвращает значение, хранящееся в маппинге по указанным ключам.

Пример использования:

// Установить значение 100 для пользователя с адресом 0x123... по ключу "balance"

contractInstance.setValue(0x123..., "balance", 100);

// Получить значение для пользователя 0x123... по ключу "balance"

uint256 value = contractInstance.getValue(0x123..., "balance"); // value будет равно 100

Use code with caution.

Применение:

Вложенные маппинги полезны для организации данных, где требуется многоуровневая структура. Например:

Хранение балансов пользователей по разным токенам:

mapping(address => mapping(string => uint256)) public balances;

balances[userAddress][tokenSymbol] = balance

Хранение настроек пользователей:

mapping(address => mapping(string => bool)) public userSettings;

userSettings[userAddress]["darkMode"] = true

Важно:

Solidity не ограничивает количество уровней вложенности маппингов, но чрезмерная вложенность может усложнить читаемость и поддержку кода.

Газ: Чтение и запись данных в маппинги потребляет газ. Старайтесь проектировать хранилище данных эффективно, чтобы минимизировать затраты на газ.

**Задача 6: Контракт с использованием структур** Напиши контракт, который использует структуры для хранения данных.

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

/\* @title Контракт для хранения данных о товарах в магазине

\* @author Bard

\* @notice Этот контракт демонстрирует использование структур для хранения данных о товарах

\*/

contract ProductStore {

/\* @notice Структура для хранения информации о товаре

\*/

struct Product {

string name; // Название товара

uint256 price; // Цена товара

uint256 quantity; // Количество товара на складе

}

/\* @notice Маппинг для хранения товаров по их ID

\* @dev Ключ - ID товара, значение - структура Product

\*/

mapping(uint256 => Product) public products;

/\* @notice Счетчик для генерации уникальных ID товаров

\*/

uint256 public nextProductId = 1;

/\* @notice Событие, которое срабатывает при добавлении нового товара

\* @param productId ID добавленного товара

\* @param name Название добавленного товара

\*/

event ProductAdded(uint256 productId, string name);

/\* @notice Функция для добавления нового товара

\* @param \_name Название товара

\* @param \_price Цена товара

\* @param \_quantity Количество товара на складе

\*/

function addProduct(string memory \_name, uint256 \_price, uint256 \_quantity) public {

products[nextProductId] = Product(\_name, \_price, \_quantity);

// ✅ Эмитируем событие после добавления товара

emit ProductAdded(nextProductId, \_name);

nextProductId++;

}

/\* @notice Функция для получения информации о товаре

\* @param \_productId ID товара

\* @return name Название товара

\* @return price Цена товара

\* @return quantity Количество товара на складе

\*/

function getProduct(uint256 \_productId) public view returns (string memory name, uint256 price, uint256 quantity) {

Product memory product = products[\_productId];

return (product.name, product.price, product.quantity);

}

}

**Пояснения:**

* **Комментарии NatSpec:** Все комментарии, начинающиеся с /\* @... \*/, соответствуют формату NatSpec и описывают контракт, структуру и функции.
* **Структура Product**: Хранит информацию о товаре: имя (name), цену (price) и количество (quantity).
* **Маппинг products**: Используется для хранения всех товаров, используя ID товара как ключ и структуру Product как значение.
* **Переменная nextProductId**: Счетчик, который увеличивается при добавлении нового товара, гарантируя уникальный ID для каждого товара.
* **Функция addProduct()**: Добавляет новый товар в маппинг products, используя nextProductId как ID и увеличивая его для следующего товара.
* **Функция getProduct()**: Возвращает информацию о товаре по его ID из маппинга products.

Этот код демонстрирует использование структур и маппингов для организации данных в смарт-контрактах, а также использование комментариев NatSpec для документирования кода.

**Задача 7: Контракт с массивом структур** Создай контракт, который использует массив структур для хранения данных.

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

/\* @title Контракт для хранения данных о сотрудниках компании

\* @author Bard

\* @notice Этот контракт демонстрирует использование массива структур для хранения данных

\*/

contract EmployeeData {

/\* @notice Структура для хранения информации о сотруднике

\*/

struct Employee {

uint256 id; // ID сотрудника

string name; // Имя сотрудника

uint256 salary; // Зарплата сотрудника

}

/\* @notice Массив для хранения данных о сотрудниках

\*/

Employee[] public employees;

/\* @notice Функция для добавления нового сотрудника

\* @param \_id ID сотрудника

\* @param \_name Имя сотрудника

\* @param \_salary Зарплата сотрудника

\*/

function addEmployee(uint256 \_id, string memory \_name, uint256 \_salary) public {

employees.push(Employee(\_id, \_name, \_salary));

}

/\* @notice Функция для получения данных о сотруднике по его индексу в массиве

\* @param \_index Индекс сотрудника в массиве

\* @return id ID сотрудника

\* @return name Имя сотрудника

\* @return salary Зарплата сотрудника

\*/

function getEmployee(uint256 \_index) public view returns (uint256 id, string memory name, uint256 salary) {

require(\_index < employees.length, "Invalid employee index");

Employee memory employee = employees[\_index];

return (employee.id, employee.name, employee.salary);

}

/\* @notice Функция для получения количества сотрудников

\* @return Количество сотрудников в массиве

\*/

function getEmployeesCount() public view returns (uint256) {

return employees.length;

}

}

**Описание:**

1. **Структура Employee**:
   * Хранит ID (id), имя (name) и зарплату (salary) сотрудника.
2. **Массив employees**:
   * Динамический массив, который хранит структуры Employee.
   * Объявлен как public, что позволяет читать его содержимое извне контракта.
3. **Функция addEmployee()**:
   * Добавляет новую структуру Employee в конец массива employees.
4. **Функция getEmployee()**:
   * Принимает индекс (\_index) элемента в массиве.
   * Возвращает ID, имя и зарплату сотрудника по указанному индексу.
   * Содержит проверку на выход за пределы массива (require(\_index < employees.length)).
5. **Функция getEmployeesCount()**:
   * Возвращает количество элементов (сотрудников) в массиве employees.

**Пример использования:**

// Создание экземпляра контракта

EmployeeData employeeData = new EmployeeData();

// Добавление сотрудников

employeeData.addEmployee(1, "Иван", 1000);

employeeData.addEmployee(2, "Мария", 1200);

// Получение данных о сотруднике с индексом 1 (Мария)

(uint256 id, string memory name, uint256 salary) = employeeData.getEmployee(1);

// Вывод: id=2, name="Мария", salary=1200

**Задача 8: Контракт с маппингом структур** Напиши контракт, который использует маппинг структур для хранения данных.

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

/\* @title Контракт для хранения данных о книгах в библиотеке

\* @author Bard

\* @notice Этот контракт демонстрирует использование маппинга структур для хранения данных

\*/contract Library {

/\* @notice Структура для хранения информации о книге

\*/

struct Book {

string title; // Название книги

string author; // Автор книги

uint256 year; // Год издания

bool available; // Доступность книги (true - доступна, false - нет)

}

/\* @notice Маппинг для хранения данных о книгах.

\* @dev Ключ - ISBN книги (строка), значение - структура Book

\*/

mapping(string => Book) public books;

/\* @notice Функция для добавления новой книги в библиотеку

\* @param \_isbn ISBN книги

\* @param \_title Название книги

\* @param \_author Автор книги

\* @param \_year Год издания

\*/

function addBook(string memory \_isbn, string memory \_title, string memory \_author, uint256 \_year) public {

books[\_isbn] = Book(\_title, \_author, \_year, true);

}

/\* @notice Функция для получения информации о книге по ISBN

\* @param \_isbn ISBN книги

\* @return title Название книги

\* @return author Автор книги

\* @return year Год издания

\* @return available Доступность книги

\*/

function getBook(string memory \_isbn) public view returns (string memory title, string memory author, uint256 year, bool available) {

Book memory book = books[\_isbn];

return (book.title, book.author, book.year, book.available);

}

/\* @notice Функция для изменения статуса доступности книги

\* @param \_isbn ISBN книги

\* @param \_available Новый статус доступности книги

\*/

function setBookAvailability(string memory \_isbn, bool \_available) public {

books[\_isbn].available = \_available;

}

}

**Описание:**

1. **Структура Book**:
   * Хранит информацию о книге: название (title), автора (author), год издания (year) и доступность (available).
2. **Маппинг books**:
   * Использует ISBN книги (строка) как ключ и структуру Book как значение.
   * Позволяет эффективно искать и получать информацию о книге по ее ISBN.
3. **Функция addBook()**:
   * Добавляет новую книгу в маппинг books с указанным ISBN и информацией.
   * При добавлении книга по умолчанию считается доступной (available = true).
4. **Функция getBook()**:
   * Возвращает информацию о книге (название, автор, год, доступность) по ее ISBN.
5. **Функция setBookAvailability()**:
   * Изменяет статус доступности книги в маппинге books.

**Пример использования:**

// Создание экземпляра контракта

Library library = new Library();

// Добавление книги

library.addBook("978-5-17-09829-2", "Война и мир", "Лев Толстой", 1869);

// Получение информации о книге

(string memory title, string memory author, uint256 year, bool available) = library.getBook("978-5-17-09829-2");

// Изменение статуса доступности книги

library.setBookAvailability("978-5-17-09829-2", false);

**Уровень 3: Продвинутые математические операции и оптимизация кода**

**Задача 9: Контракт с использованием библиотеки SafeMath** Создай контракт, который использует библиотеку SafeMath для безопасных математических операций.

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Импортируем библиотеку SafeMath

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/math/SafeMath.sol";

/\* @title Контракт для демонстрации использования SafeMath

\* @author Bard

\* @notice Этот контракт использует SafeMath для безопасных математических операций

\*/

contract SafeMathExample {

// Используем SafeMath для uint256

using SafeMath for uint256;

/\* @notice Переменная для хранения баланса

\*/

uint256 public balance;

/\* @notice Функция для увеличения баланса

\* @param \_amount Количество, на которое нужно увеличить баланс

\*/

function deposit(uint256 \_amount) public {

balance = balance.add(\_amount);

}

/\* @notice Функция для уменьшения баланса

\* @param \_amount Количество, на которое нужно уменьшить баланс

\*/

function withdraw(uint256 \_amount) public {

balance = balance.sub(\_amount);

}

}

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).Solidity

**Описание:**

1. **Импорт SafeMath:**
   * import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/math/SafeMath.sol";
   * Импортирует библиотеку SafeMath из репозитория OpenZeppelin.
2. **Использование SafeMath для uint256:**
   * using SafeMath for uint256;
   * Применяет функции SafeMath ко всем переменным типа uint256 в этом контракте.
3. **Функции deposit() и withdraw():**
   * Используют функции add() и sub() из SafeMath для безопасного увеличения и уменьшения баланса.
   * SafeMath предотвращает переполнение и обнуление значений, которые могут возникнуть при обычных математических операциях.

**Пример использования:**

// Создание экземпляра контракта

SafeMathExample safeMath = new SafeMathExample();

// Увеличение баланса на 10

safeMath.deposit(10);

// Уменьшение баланса на 5

safeMath.withdraw(5);

content\_copyUse code [with caution](https://support.google.com/legal/answer/13505487).

**Преимущества использования SafeMath:**

* **Предотвращение переполнения:** add() и sub() проверяют, не приведет ли операция к переполнению, и вызывают ошибку, если это так.
* **Защита от обнуления:** sub() гарантирует, что результат вычитания не будет меньше нуля.
* **Улучшение читаемости:** Использование add() и sub() вместо + и - делает код более читаемым и понятным, выделяя, что используются безопасные операции.

**Задача 10: Контракт с функцией возведения в степень** Напиши контракт, который содержит функцию для возведения числа в степень.

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

/// @title Контракт для вычисления степени числа

/// @author Claude AI

/// @notice Этот контракт предоставляет функцию для вычисления степени числа

/// @dev Все вызовы функций в настоящее время реализованы без побочных эффектов

contract PowerCalculator {

/// @notice Вычисляет степень числа

/// @dev Эта функция использует простой итеративный подход для вычисления степени

/// @param base Основание степени

/// @param exponent Показатель степени (должен быть неотрицательным)

/// @return result Результат возведения base в степень exponent

/// @custom:caution Эта функция может потреблять много газа для больших показателей степени

function calculatePower(uint256 base, uint256 exponent) public pure returns (uint256 result) {

require(exponent >= 0, "Показатель степени должен быть неотрицательным");

if (exponent == 0) {

return 1;

}

result = 1;

for (uint256 i = 0; i < exponent; i++) {

result \*= base;

}

return result;

}

}

**Задача 11: Контракт с функцией факториала** Создай контракт, который содержит функцию для вычисления факториала числа.

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт с функцией факториала

contract Factorial {

// Функция для вычисления факториала числа

function factorial(uint256 number) public pure returns (uint256) {

require(number >= 0, "Number must be non-negative");

if (number == 0) {

return 1;

} else {

uint256 result = 1;

for (uint256 i = 1; i <= number; i++) {

result \*= i;

}

return result;

}

}

}

**Задача 12: Контракт с оптимизацией хранения данных** Напиши контракт, который оптимизирует хранение данных с использованием битовых полей.

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт с оптимизацией хранения данных с использованием битовых полей

contract BitwiseStorage {

uint256 public data;

// Функция для установки значения в битовое поле

function setBit(uint256 position) public {

data |= (1 << position);

}

// Функция для сброса значения в битовом поле

function clearBit(uint256 position) public {

data &= ~(1 << position);

}

// Функция для проверки значения в битовом поле

function checkBit(uint256 position) public view returns (bool) {

return (data & (1 << position)) != 0;

}

}

**Уровень 4: Расширенные темы**

**Задача 13: Контракт для голосования с делегированием** Создай контракт для голосования, где можно делегировать свои голоса другому участнику.

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт для голосования с делегированием

contract DelegatedVoting {

struct Voter {

uint256 weight;

bool voted;

address delegate;

uint256 vote;

}

mapping(address => Voter) public voters;

uint256[] public proposals;

// Конструктор для создания контракта

constructor(uint256 proposalCount) {

proposals.length = proposalCount;

}

// Функция для предоставления права голоса

function giveRightToVote(address voter) public {

require(!voters[voter].voted, "Already voted");

require(voters[voter].weight == 0, "Already has right to vote");

voters[voter].weight = 1;

}

// Функция для делегирования голоса

function delegate(address to) public {

Voter storage sender = voters[msg.sender];

require(!sender.voted, "Already voted");

require(to != msg.sender, "Cannot delegate to self");

while (voters[to].delegate != address(0)) {

to = voters[to].delegate;

require(to != msg.sender, "Found loop in delegation");

}

sender.voted = true;

sender.delegate = to;

Voter storage delegate\_ = voters[to];

if (delegate\_.voted) {

proposals[delegate\_.vote] += sender.weight;

} else {

delegate\_.weight += sender.weight;

}

}

// Функция для голосования за предложение

function vote(uint256 proposal) public {

Voter storage sender = voters[msg.sender];

require(sender.weight != 0, "Has no right to vote");

require(!sender.voted, "Already voted");

sender.voted = true;

sender.vote = proposal;

proposals[proposal] += sender.weight;

}

// Функция для подсчета голосов

function winningProposal() public view returns (uint256 winningProposal\_) {

uint256 winningVoteCount = 0;

for (uint256 p = 0; p < proposals.length; p++) {

if (proposals[p] > winningVoteCount) {

winningVoteCount = proposals[p];

winningProposal\_ = p;

}

}

}

}

**Задача 14: Контракт для аукциона с возвратом средств** Создай контракт для аукциона, который позволяет участникам вернуть свои ставки, если они не выиграли.

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт для аукциона с возвратом средств

contract Auction {

address payable public beneficiary;

uint256 public auctionEndTime;

address public highestBidder;

uint256 public highestBid;

mapping(address => uint256) pendingReturns;

bool ended;

// События для отслеживания хода аукциона

event HighestBidIncreased(address bidder, uint256 amount);

event AuctionEnded(address winner, uint256 amount);

// Конструктор для создания аукциона

constructor(uint256 biddingTime, address payable beneficiaryAddress) {

beneficiary = beneficiaryAddress;

auctionEndTime = block.timestamp + biddingTime;

}

// Функция для участия в аукционе

function bid() public payable {

require(block.timestamp <= auctionEndTime, "Auction already ended");

require(msg.value > highestBid, "There already is a higher bid");

if (highestBid != 0) {

pendingReturns[highestBidder] += highestBid;

}

highestBidder = msg.sender;

highestBid = msg.value;

emit HighestBidIncreased(msg.sender, msg.value);

}

// Функция для вывода средств, если ставка не выиграла

function withdraw() public returns (bool) {

uint256 amount = pendingReturns[msg.sender];

if (amount > 0) {

pendingReturns[msg.sender] = 0;

if (!payable(msg.sender).send(amount)) {

pendingReturns[msg.sender] = amount;

return false;

}

}

return true;

}

// Функция для завершения аукциона

function endAuction() public {

require(block.timestamp >= auctionEndTime, "Auction not yet ended");

require(!ended, "Auction end already called");

ended = true;

emit AuctionEnded(highestBidder, highestBid);

beneficiary.transfer(highestBid);

}

}

**Задача 15: Контракт для управления доступом на основе ролей** Напиши контракт, который управляет доступом к функциям на основе ролей.

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "@openzeppelin/contracts/access/AccessControl.sol";

// Контракт для управления доступом на основе ролей

contract RoleBasedAccess is AccessControl {

bytes32 public constant ADMIN\_ROLE = keccak256("ADMIN\_ROLE");

bytes32 public constant USER\_ROLE = keccak256("USER\_ROLE");

// Конструктор для задания первоначальных ролей

constructor() {

\_setupRole(ADMIN\_ROLE, msg.sender);

\_setRoleAdmin(USER\_ROLE, ADMIN\_ROLE);

}

// Функция для назначения роли пользователя

function addUser(address account) public {

require(hasRole(ADMIN\_ROLE, msg.sender), "Caller is not an admin");

grantRole(USER\_ROLE, account);

}

// Функция, доступная только пользователям

function userFunction() public view {

require(hasRole(USER\_ROLE, msg.sender), "Caller is not a user");

// Действия для пользователей

}

// Функция, доступная только администраторам

function adminFunction() public view {

require(hasRole(ADMIN\_ROLE, msg.sender), "Caller is not an admin");

// Действия для администраторов

}

}

**Уровень 5: Расширенные темы и практики**

**Задача 16: Контракт с ограниченным количеством вызовов функции** Создай контракт, который ограничивает количество вызовов функции.

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт с ограниченным количеством вызовов функции

contract LimitedCalls {

uint256 public callCount;

uint256 public maxCalls;

// Конструктор для задания максимального количества вызовов

constructor(uint256 \_maxCalls) {

maxCalls = \_maxCalls;

}

// Функция, количество вызовов которой ограничено

function limitedFunction() public {

require(callCount < maxCalls, "Max calls exceeded");

callCount++;

}

}

**Задача 17: Контракт с отслеживанием изменения состояния** Напиши контракт, который отслеживает изменение состояния и генерирует события при изменении.

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт с отслеживанием изменения состояния

contract StateTracking {

enum State { Created, InProgress, Completed }

State public state;

// Событие для отслеживания изменения состояния

event StateChanged(State newState);

// Функция для изменения состояния

function changeState(State newState) public {

state = newState;

emit StateChanged(newState);

}

}

**Задача 18: Контракт с функцией для работы с большим массивом** Создай контракт, который содержит функцию для обработки большого массива с использованием циклов.

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт с функцией для работы с большим массивом

contract LargeArrayProcessing {

uint256[] public largeArray;

// Функция для добавления значений в массив

function addValues(uint256[] memory values) public {

for (uint256 i = 0; i < values.length; i++) {

largeArray.push(values[i]);

}

}

// Функция для суммирования всех значений в массиве

function sumValues() public view returns (uint256) {

uint256 sum = 0;

for (uint256 i = 0; i < largeArray.length; i++) {

sum += largeArray[i];

}

return sum;

}

}

**Задача 19: Контракт с оптимизацией использования газа** Напиши контракт, который использует битовые операции для оптимизации использования газа.

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт с оптимизацией использования газа с помощью битовых операций

contract GasOptimized {

uint256 public data;

// Функция для установки значения с использованием битовых операций

function setBits(uint256 position, bool value) public {

if (value) {

data |= (1 << position);

} else {

data &= ~(1 << position);

}

}

// Функция для получения значения с использованием битовых операций

function getBit(uint256 position) public view returns (bool) {

return (data & (1 << position)) != 0;

}

}

**Задача 20: Контракт с использованием библиотеки для работы с большими числами** Создай контракт, который использует библиотеку BigNumber для работы с большими числами.

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "@openzeppelin/contracts/utils/math/SafeMath.sol";

// Контракт с использованием библиотеки BigNumber

contract BigNumberExample {

using SafeMath for uint256;

uint256 public largeNumber;

// Функция для установки большого числа

function setLargeNumber(uint256 number) public {

largeNumber = number;

}

// Функция для умножения большого числа

function multiplyLargeNumber(uint256 multiplier) public {

largeNumber = largeNumber.mul(multiplier);

}

}

**Уровень 6: Продвинутые функции и оптимизация**

**Задача 21: Контракт с функцией для вычисления корня квадратного** Создай контракт, который содержит функцию для вычисления корня квадратного числа.

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт с функцией для вычисления корня квадратного

contract SquareRoot {

// Функция для вычисления корня квадратного числа

function sqrt(uint256 x) public pure returns (uint256) {

uint256 z = (x + 1) / 2;

uint256 y = x;

while (z < y) {

y = z;

z = (x / z + z) / 2;

}

return y;

}

}

**Задача 22: Контракт с функцией для вычисления среднего значения** Напиши контракт, который содержит функцию для вычисления среднего значения массива чисел.

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт с функцией для вычисления среднего значения

contract Average {

// Функция для вычисления среднего значения массива чисел

function calculateAverage(uint256[] memory numbers) public pure returns (uint256) {

uint256 sum = 0;

for (uint256 i = 0; i < numbers.length; i++) {

sum += numbers[i];

}

return sum / numbers.length;

}

}

**Задача 23: Контракт с функцией для сортировки массива** Создай контракт, который содержит функцию для сортировки массива чисел.

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт с функцией для сортировки массива чисел

contract SortArray {

// Функция для сортировки массива чисел

function sort(uint256[] memory array) public pure returns (uint256[] memory) {

uint256 n = array.length;

for (uint256 i = 0; i < n; i++) {

for (uint256 j = 0; j < n - 1; j++) {

if (array[j] > array[j + 1]) {

uint256 temp = array[j];

array[j] = array[j + 1];

array[j + 1] = temp;

}

}

}

return array;

}

}

**Задача 24: Контракт с использованием хэш-таблиц** Напиши контракт, который использует хэш-таблицы для хранения и поиска данных.

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт с использованием хэш-таблиц

contract HashTable {

struct Entry {

uint256 key;

uint256 value;

}

mapping(uint256 => Entry) public table;

// Функция для добавления записи в хэш-таблицу

function addEntry(uint256 key, uint256 value) public {

table[key] = Entry(key, value);

}

// Функция для получения значения по ключу

function getValue(uint256 key) public view returns (uint256) {

return table[key].value;

}

}

**Уровень 7: Продвинутые структуры данных и алгоритмы**

**Задача 25: Контракт с функцией для работы с деревом** Создай контракт, который использует дерево для хранения данных и содержит функции для добавления и поиска данных.

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт с функцией для работы с деревом

contract BinaryTree {

struct Node {

uint256 value;

Node left;

Node right;

}

Node public root;

// Функция для добавления узла в дерево

function addNode(uint256 value) public {

root = addNodeRec(root, value);

}

// Вспомогательная рекурсивная функция для добавления узла

function addNodeRec(Node storage node, uint256 value) internal returns (Node storage) {

if (node == Node(0)) {

return Node(value, Node(0), Node(0));

}

if (value < node.value) {

node.left = addNodeRec(node.left, value);

} else {

node.right = addNodeRec(node.right, value);

}

return node;

}

// Функция для поиска узла в дереве

function findNode(uint256 value) public view returns (bool) {

return findNodeRec(root, value);

}

// Вспомогательная рекурсивная функция для поиска узла

function findNodeRec(Node storage node, uint256 value) internal view returns (bool) {

if (node == Node(0)) {

return false;

}

if (node.value == value) {

return true;

} else if (value < node.value) {

return findNodeRec(node.left, value);

} else {

return findNodeRec(node.right, value);

}

}

}

**Задача 26: Контракт с функцией для работы с графом** Напиши контракт, который использует граф для хранения данных и содержит функции для добавления и поиска вершин.

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт с функцией для работы с графом

contract Graph {

struct Node {

uint256 value;

uint256[] neighbors;

}

mapping(uint256 => Node) public nodes;

// Функция для добавления вершины в граф

function addNode(uint256 value) public {

nodes[value] = Node(value, new uint256[](0));

}

// Функция для добавления ребра между вершинами

function addEdge(uint256 from, uint256 to) public {

nodes[from].neighbors.push(to);

nodes[to].neighbors.push(from);

}

// Функция для поиска вершины в графе

function findNode(uint256 value) public view returns (bool) {

return nodes[value].value != 0;

}

// Функция для получения соседей вершины

function getNeighbors(uint256 value) public view returns (uint256[] memory) {

return nodes[value].neighbors;

}

}

**Задача 27: Контракт с функцией для работы с очередью** Создай контракт, который использует очередь для хранения данных и содержит функции для добавления и удаления данных.

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт с функцией для работы с очередью

contract Queue {

uint256[] public queue;

uint256 public front;

uint256 public rear;

// Конструктор для инициализации очереди

constructor() {

front = 0;

rear = 0;

}

// Функция для добавления элемента в очередь

function enqueue(uint256 value) public {

queue.push(value);

rear++;

}

// Функция для удаления элемента из очереди

function dequeue() public returns (uint256) {

require(front < rear, "Queue is empty");

uint256 value = queue[front];

front++;

return value;

}

// Функция для получения размера очереди

function size() public view returns (uint256) {

return rear - front;

}

}

**Задача 28: Контракт с функцией для работы с стеком** Напиши контракт, который использует стек для хранения данных и содержит функции для добавления и удаления данных.

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт с функцией для работы с стеком

contract Stack {

uint256[] public stack;

// Функция для добавления элемента в стек

function push(uint256 value) public {

stack.push(value);

}

// Функция для удаления элемента из стека

function pop() public returns (uint256) {

require(stack.length > 0, "Stack is empty");

uint256 value = stack[stack.length - 1];

stack.pop();

return value;

}

// Функция для получения размера стека

function size() public view returns (uint256) {

return stack.length;

}

}

**Задача 29: Контракт с функцией для работы с кучей** Создай контракт, который использует кучу для хранения данных и содержит функции для добавления и удаления данных.

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт с функцией для работы с кучей

contract Heap {

uint256[] public heap;

// Функция для добавления элемента в кучу

function insert(uint256 value) public {

heap.push(value);

uint256 i = heap.length - 1;

while (i > 0) {

uint256 parent = (i - 1) / 2;

if (heap[parent] >= heap[i]) {

break;

}

(heap[parent], heap[i]) = (heap[i], heap[parent]);

i = parent;

}

}

// Функция для удаления элемента из кучи

function extractMax() public returns (uint256) {

require(heap.length > 0, "Heap is empty");

uint256 max = heap[0];

heap[0] = heap[heap.length - 1];

heap.pop();

uint256 i = 0;

while (true) {

uint256 left = 2 \* i + 1;

uint256 right = 2 \* i + 2;

uint256 largest = i;

if (left < heap.length && heap[left] > heap[largest]) {

largest = left;

}

if (right < heap.length && heap[right] > heap[largest]) {

largest = right;

}

if (largest == i) {

break;

}

(heap[i], heap[largest]) = (heap[largest], heap[i]);

i = largest;

}

return max;

}

}

**Задача 30: Контракт с функцией для работы с хэш-таблицами с коллизиями** Напиши контракт, который использует хэш-таблицы с обработкой коллизий для хранения данных.

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт с функцией для работы с хэш-таблицами с коллизиями

contract HashTableWithCollisions {

struct Entry {

uint256 key;

uint256 value;

Entry next;

}

mapping(uint256 => Entry) public table;

uint256 public size;

// Конструктор для инициализации размера таблицы

constructor(uint256 \_size) {

size = \_size;

}

// Функция для вычисления хэша

function hash(uint256 key) public view returns (uint256) {

return key % size;

}

// Функция для добавления записи в хэш-таблицу

function addEntry(uint256 key, uint256 value) public {

uint256 index = hash(key);

Entry storage entry = table[index];

while (entry.key != 0) {

entry = entry.next;

}

entry.key = key;

entry.value = value;

entry.next = Entry(0, 0, Entry(0));

}

// Функция для получения значения по ключу

function getValue(uint256 key) public view returns (uint256) {

uint256 index = hash(key);

Entry storage entry = table[index];

while (entry.key != key) {

entry = entry.next;

require(entry.key != 0, "Key not found");

}

return entry.value;

}

}

**Задача 25: Контракт для реализации события с передачей структуры**

Создай контракт, который использует событие для передачи структуры данных.

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт для реализации события с передачей структуры

contract EventWithStruct {

struct Person {

string name;

uint age;

}

event NewPersonAdded(Person person);

function addPerson(string memory name, uint age) public {

Person memory newPerson = Person(name, age);

emit NewPersonAdded(newPerson);

}

}

**Задача 26: Контракт для кодирования и декодирования данных**

Создай контракт, который использует abi.encode и abi.decode для кодирования и декодирования данных.

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт для кодирования и декодирования данных

contract EncodeDecode {

function encodeData(string memory name, uint age) public pure returns (bytes memory) {

return abi.encode(name, age);

}

function decodeData(bytes memory data) public pure returns (string memory, uint) {

return abi.decode(data, (string, uint));

}

}

**Задача 27: Контракт для использования фиксированного размера массива**

Создай контракт, который использует массив фиксированного размера типа bytes32.

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт для использования фиксированного размера массива

contract FixedSizeArray {

bytes32[5] public data;

function setData(uint index, bytes32 value) public {

require(index < data.length, "Index out of bounds");

data[index] = value;

}

function getData(uint index) public view returns (bytes32) {

require(index < data.length, "Index out of bounds");

return data[index];

}

}

**Задача 28: Контракт для реализации мультиподписи**

Создай контракт, который реализует механизм мультиподписи для выполнения транзакций.

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт для реализации мультиподписи

contract MultiSig {

address[] public owners;

uint public requiredSignatures;

mapping(address => bool) public isOwner;

mapping(uint => mapping(address => bool)) public approvals;

struct Transaction {

address to;

uint value;

bool executed;

}

Transaction[] public transactions;

constructor(address[] memory \_owners, uint \_requiredSignatures) {

require(\_owners.length > 0, "Owners required");

require(\_requiredSignatures > 0 && \_requiredSignatures <= \_owners.length, "Invalid required signatures");

for (uint i = 0; i < \_owners.length; i++) {

isOwner[\_owners[i]] = true;

}

owners = \_owners;

requiredSignatures = \_requiredSignatures;

}

function submitTransaction(address to, uint value) public {

require(isOwner[msg.sender], "Not owner");

transactions.push(Transaction(to, value, false));

}

function approveTransaction(uint txIndex) public {

require(isOwner[msg.sender], "Not owner");

require(txIndex < transactions.length, "Invalid transaction");

require(!approvals[txIndex][msg.sender], "Already approved");

approvals[txIndex][msg.sender] = true;

}

function executeTransaction(uint txIndex) public {

require(txIndex < transactions.length, "Invalid transaction");

Transaction storage transaction = transactions[txIndex];

uint approvalCount = 0;

for (uint i = 0; i < owners.length; i++) {

if (approvals[txIndex][owners[i]]) {

approvalCount++;

}

}

require(approvalCount >= requiredSignatures, "Not enough approvals");

require(!transaction.executed, "Transaction already executed");

transaction.executed = true;

payable(transaction.to).transfer(transaction.value);

}

}

**Задача 29: Контракт для реализации системы голосования с делегированием**

Создай контракт, который реализует систему голосования с возможностью делегирования голосов.

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт для реализации системы голосования с делегированием

contract DelegatedVoting {

struct Voter {

uint weight;

bool voted;

address delegate;

uint vote;

}

struct Proposal {

string name;

uint voteCount;

}

address public chairperson;

mapping(address => Voter) public voters;

Proposal[] public proposals;

constructor(string[] memory proposalNames) {

chairperson = msg.sender;

voters[chairperson].weight = 1;

for (uint i = 0; i < proposalNames.length; i++) {

proposals.push(Proposal({name: proposalNames[i], voteCount: 0}));

}

}

function giveRightToVote(address voter) public {

require(msg.sender == chairperson, "Only chairperson can give right to vote");

require(!voters[voter].voted, "The voter already voted");

require(voters[voter].weight == 0, "The voter already has right to vote");

voters[voter].weight = 1;

}

function delegate(address to) public {

Voter storage sender = voters[msg.sender];

require(!sender.voted, "You already voted");

while (voters[to].delegate != address(0) && voters[to].delegate != msg.sender) {

to = voters[to].delegate;

}

require(to != msg.sender, "Found loop in delegation");

sender.voted = true;

sender.delegate = to;

Voter storage delegate\_ = voters[to];

if (delegate\_.voted) {

proposals[delegate\_.vote].voteCount += sender.weight;

} else {

delegate\_.weight += sender.weight;

}

}

function vote(uint proposal) public {

Voter storage sender = voters[msg.sender];

require(sender.weight != 0, "Has no right to vote");

require(!sender.voted, "Already voted");

sender.voted = true;

sender.vote = proposal;

proposals[proposal].voteCount += sender.weight;

}

function winningProposal() public view returns (uint winningProposal\_) {

uint winningVoteCount = 0;

for (uint p = 0; p < proposals.length; p++) {

if (proposals[p].voteCount > winningVoteCount) {

winningVoteCount = proposals[p].voteCount;

winningProposal\_ = p;

}

}

}

function winnerName() public view returns (string memory) {

return proposals[winningProposal()].name;

}

}

**Задача 30: Контракт для реализации дерева Меркла**

Создай контракт, который реализует дерево Меркла для проверки целостности данных.

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "@openzeppelin/contracts/utils/cryptography/MerkleProof.sol";

// Контракт для реализации дерева Меркла

contract MerkleTree {

bytes32 public merkleRoot;

// Функция для установки корня Меркла

function setMerkleRoot(bytes32 root) public {

merkleRoot = root;

}

// Функция для проверки включения элемента в дерево Меркла

function verify(bytes32[] memory proof, bytes32 leaf) public view returns (bool) {

return MerkleProof.verify(proof, merkleRoot, leaf);

}

}

**Задача 25: Контракт для продажи токенов с использованием SafeMath**

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "@openzeppelin/contracts/utils/math/SafeMath.sol";

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

// Контракт для продажи токенов с использованием SafeMath

contract TokenSale {

using SafeMath for uint256;

IERC20 public token;

address public owner;

uint256 public price; // Цена одного токена в wei

constructor(IERC20 \_token, uint256 \_price) {

token = \_token;

price = \_price;

owner = msg.sender;

}

modifier onlyOwner() {

require(msg.sender == owner, "Not the owner");

\_;

}

// Функция для покупки токенов

function buyTokens(uint256 amount) public payable {

uint256 cost = amount.mul(price); // Стоимость покупки в wei

require(msg.value >= cost, "Not enough ether sent");

uint256 balance = token.balanceOf(address(this));

require(balance >= amount, "Not enough tokens in contract");

// Перевод токенов покупателю

token.transfer(msg.sender, amount);

// Возврат лишнего эфира

if (msg.value > cost) {

payable(msg.sender).transfer(msg.value.sub(cost));

}

}

// Функция для вывода эфира владельцем контракта

function withdraw() public onlyOwner {

payable(owner).transfer(address(this).balance);

}

}

**Задача 26: Контракт для управления реферальной программой**

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт для управления реферальной программой

contract ReferralProgram {

mapping(address => address) public referrals;

mapping(address => uint256) public rewards;

uint256 public rewardAmount = 0.01 ether;

// Функция для регистрации рефералов

function registerReferral(address referrer) public {

require(referrals[msg.sender] == address(0), "Already referred");

require(referrer != msg.sender, "Cannot refer yourself");

referrals[msg.sender] = referrer;

}

// Функция для начисления наград

function rewardReferral(address referrer) public payable {

require(msg.value == rewardAmount, "Incorrect reward amount");

require(referrals[referrer] != address(0), "Referrer has no referrals");

rewards[referrer] += msg.value;

}

// Функция для вывода наград

function withdrawRewards() public {

uint256 reward = rewards[msg.sender];

require(reward > 0, "No rewards available");

rewards[msg.sender] = 0;

payable(msg.sender).transfer(reward);

}

// Функция для просмотра информации о рефералах

function getReferralInfo(address user) public view returns (address, uint256) {

return (referrals[user], rewards[user]);

}

}

**Задача 27: Контракт для управления членством с периодическими взносами**

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт для управления членством с периодическими взносами

contract Membership {

struct Member {

uint256 expiry;

bool active;

}

mapping(address => Member) public members;

uint256 public membershipFee = 0.1 ether;

uint256 public membershipDuration = 30 days;

// Функция для покупки членства

function buyMembership() public payable {

require(msg.value == membershipFee, "Incorrect membership fee");

Member storage member = members[msg.sender];

if (member.active) {

member.expiry += membershipDuration;

} else {

member.expiry = block.timestamp + membershipDuration;

member.active = true;

}

}

// Функция для проверки статуса членства

function checkMembership(address user) public view returns (bool) {

return members[user].active && block.timestamp < members[user].expiry;

}

// Функция для продления членства

function renewMembership() public payable {

require(msg.value == membershipFee, "Incorrect membership fee");

require(members[msg.sender].active, "Not a member");

members[msg.sender].expiry += membershipDuration;

}

// Функция для отмены членства

function cancelMembership() public {

require(members[msg.sender].active, "Not a member");

members[msg.sender].active = false;

}

}

**Задача 28: Контракт для реализации системы депозитов с блокировкой средств**

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт для реализации системы депозитов с блокировкой средств

contract DepositLock {

struct Deposit {

uint256 amount;

uint256 unlockTime;

}

mapping(address => Deposit) public deposits;

// Функция для внесения депозита с указанием времени блокировки

function deposit(uint256 lockTime) public payable {

require(msg.value > 0, "Deposit amount must be greater than zero");

deposits[msg.sender] = Deposit({

amount: msg.value,

unlockTime: block.timestamp + lockTime

});

}

// Функция для вывода депозита после завершения времени блокировки

function withdraw() public {

Deposit memory userDeposit = deposits[msg.sender];

require(userDeposit.amount > 0, "No deposit found");

require(block.timestamp >= userDeposit.unlockTime, "Deposit is still locked");

uint256 amount = userDeposit.amount;

deposits[msg.sender].amount = 0;

payable(msg.sender).transfer(amount);

}

// Функция для получения информации о депозите

function getDepositInfo(address user) public view returns (uint256, uint256) {

Deposit memory userDeposit = deposits[user];

return (userDeposit.amount, userDeposit.unlockTime);

}

}

**Задача 29: Контракт для управления аукционами с использованием гибких ставок**

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт для управления аукционами с использованием гибких ставок

contract FlexibleAuction {

struct Bid {

address bidder;

uint256 amount;

}

Bid public highestBid;

address public seller;

bool public ended;

event NewHighestBid(address bidder, uint256 amount);

event AuctionEnded(address winner, uint256 amount);

constructor() {

seller = msg.sender;

}

// Функция для размещения ставки

function placeBid() public payable {

require(!ended, "Auction already ended");

require(msg.value > highestBid.amount, "There already is a higher bid");

if (highestBid.amount != 0) {

// Возврат предыдущей ставки

payable(highestBid.bidder).transfer(highestBid.amount);

}

highestBid = Bid({

bidder: msg.sender,

amount: msg.value

});

emit NewHighestBid(msg.sender, msg.value);

}

// Функция для завершения аукциона

function endAuction() public {

require(msg.sender == seller, "You are not the seller");

require(!ended, "Auction already ended");

ended = true;

emit AuctionEnded(highestBid.bidder, highestBid.amount);

payable(seller).transfer(highestBid.amount);

}

// Функция для получения информации о текущей самой высокой ставке

function getHighestBid() public view returns (address, uint256) {

return (highestBid.bidder, highestBid.amount);

}

}

**Задача 30: Контракт для создания простого DAO**

solidity

Копировать код

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

// Контракт для создания простого DAO

contract SimpleDAO {

struct Proposal {

string description;

uint256 voteCount;

}

Proposal[] public proposals;

mapping(address => bool) public members;

mapping(uint256 => mapping(address => bool)) public votes;

address public admin;

constructor() {

admin = msg.sender;

members[admin] = true;

}

modifier onlyMember() {

require(members[msg.sender], "Not a member");

\_;

}

modifier onlyAdmin() {

require(msg.sender == admin, "Not an admin");

\_;

}

// Функция для добавления нового члена

function addMember(address member) public onlyAdmin {

members[member] = true;

}

// Функция для создания нового предложения

function createProposal(string memory description) public onlyMember {

proposals.push(Proposal({

description: description,

voteCount: 0

}));

}

// Функция для голосования за предложение

function vote(uint256 proposalIndex) public onlyMember {

require(proposalIndex < proposals.length, "Invalid proposal");

require(!votes[proposalIndex][msg.sender], "Already voted");

proposals[proposalIndex].voteCount++;

votes[proposalIndex][msg.sender] = true;

}

// Функция для получения информации о предложении

function getProposal(uint256 proposalIndex) public view returns (string memory, uint256) {

require(proposalIndex < proposals.length, "Invalid proposal");

Proposal memory proposal = proposals[proposalIndex];

return (proposal.description, proposal.voteCount);

}

}