# ПРИЛОЖЕНИЕ 1

# 1.1 Листинг кода

// SPDX-License-Identifrier: MIT

pragma solidity ^0.8.0;

contract GradeBook {

    struct GradeRecord {

        string instituteName;   // Название института

        string subject;         // Название предмета

        uint8 semester;         // Номер семестра

        string teacherName;     // ФИО преподавателя

        string studentName;     // ФИО студента

        uint256 studentID;      // Номер зачётки студента

        uint8 grade;            // Оценка

        uint256 date;           // Дата

    }

    mapping(bytes32 => GradeRecord) private gradeRecords;

    // Функция для добавления записи об оценке

    function addGradeRecord(

        string memory \_instituteName,

        string memory \_subject,

        uint8 \_semester,

        string memory \_teacherName,

        string memory \_studentName,

        uint256 \_studentID,

        uint8 \_grade,

        uint256 \_date

    ) public {

        // Генерируем хеш для записи

        bytes32 hash = generateHash(\_studentID, \_subject, \_semester);

        // Сохраняем запись в маппинге

        gradeRecords[hash] = GradeRecord(

            \_instituteName,

            \_subject,

            \_semester,

            \_teacherName,

            \_studentName,

            \_studentID,

            \_grade,

            \_date

        );

    }

    // Функция для получения записи об оценке

    function getGradeRecord(

        uint256 \_studentID,

        string memory \_subject,

        uint8 \_semester

    ) public view returns (

        string memory instituteName,

        string memory subject,

        uint8 semester,

        string memory teacherName,

        string memory studentName,

        uint256 studentID,

        uint8 grade,

        uint256 date

    ) {

        // Генерируем хеш для записи

        bytes32 hash = generateHash(\_studentID, \_subject, \_semester);

        // Получаем запись из маппинга

        GradeRecord storage record = gradeRecords[hash];

        // Возвращаем отдельные поля записи

        return (

            record.instituteName,

            record.subject,

            record.semester,

            record.teacherName,

            record.studentName,

            record.studentID,

            record.grade,

            record.date

        );

    }

    // Функция для генерации уникального хеша для записи об оценке

    function generateHash(

        uint256 \_studentID,

        string memory \_subject,

        uint8 \_semester

    ) private pure returns (bytes32) {

        // Конкатенируем поля и вычисляем хеш

        return keccak256(abi.encodePacked(\_studentID, \_subject, \_semester));

    }

    // Функция для добавления нескольких записей об оценках из XML

    function addGradeRecordsFromXML(string memory \_xmlData) public {

        // Разбиваем XML на отдельные записи

        string[] memory records = splitXMLRecords(\_xmlData);

        // Проверяем наличие записей

        if (records.length == 0) {

            revert("No grade records found in XML.");

        }

        // Обрабатываем каждую запись и добавляем ее в блокчейн

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

            GradeRecord memory record = parseXMLRecord(records[i]);

            addGradeRecord(

                record.instituteName,

                record.subject,

                record.semester,

                record.teacherName,

                record.studentName,

                record.studentID,

                record.grade,

                record.date

            );

        }

    }

    // Функция для парсинга записи об оценке из XML

    function parseXMLRecord(string memory \_xmlData) internal pure returns (GradeRecord memory) {

        GradeRecord memory record;

        record.instituteName = getValueFromTag(\_xmlData, "institute");

        record.subject = getValueFromTag(\_xmlData, "subject");

        record.semester = uint8(parseInt(getValueFromTag(\_xmlData, "semester")));

        record.teacherName = getValueFromTag(\_xmlData, "teacher");

        record.studentName = getValueFromTag(\_xmlData, "fullName");

        record.studentID = parseInt(getValueFromTag(\_xmlData, "studentNumber"));

        record.grade = uint8(parseInt(getValueFromTag(\_xmlData, "grade")));

        record.date = parseInt(getValueFromTag(\_xmlData, "date"));

        return record;

    }

    // Функция для разделения XML на отдельные записи

    function splitXMLRecords(string memory \_xmlData) internal pure returns (string[] memory) {

        // Разделитель записей

        string memory recordDelimiter = "</gradeRecord>";

        // Разделяем XML на отдельные записи

        string[] memory records = splitString(\_xmlData, recordDelimiter);

        // Удаляем пустые записи

        uint256 validRecordCount = 0;

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

            if (bytes(records[i]).length > 0) {

                records[validRecordCount] = records[i];

                validRecordCount++;

            }

        }

        // Создаем новый массив с валидными записями

        string[] memory validRecords = new string[](validRecordCount);

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

            validRecords[i] = records[i];

        }

        return validRecords;

    }

    // Функция для получения значения из тега XML

    function getValueFromTag(string memory \_xmlData, string memory \_tag) internal pure returns (string memory) {

        string memory startTag = string(abi.encodePacked("<", \_tag, ">"));

        string memory endTag = string(abi.encodePacked("</", \_tag, ">"));

        uint256 startPos = indexOf(\_xmlData, startTag) + bytes(startTag).length;

        uint256 endPos = indexOf(\_xmlData, endTag);

        return substring(\_xmlData, startPos, endPos);

    }

    // Функция для разделения строки на массив по разделителю

    function splitString(string memory \_string, string memory \_delimiter) internal pure returns (string[] memory) {

        bytes memory stringBytes = bytes(\_string);

        bytes memory delimiterBytes = bytes(\_delimiter);

        uint256 delimiterCount = countOccurrences(\_string, \_delimiter);

        string[] memory parts = new string[](delimiterCount + 1);

        uint256 partIndex = 0;

        uint256 startPos = 0;

        for (uint256 i = 0; i < stringBytes.length - delimiterBytes.length + 1; i++) {

            bool isDelimiter = true;

            for (uint256 j = 0; j < delimiterBytes.length; j++) {

                if (stringBytes[i + j] != delimiterBytes[j]) {

                    isDelimiter = false;

                    break;

                }

            }

            if (isDelimiter) {

                parts[partIndex] = substring(\_string, startPos, i);

                partIndex++;

                startPos = i + delimiterBytes.length;

            }

        }

        // Добавляем последнюю часть строки

        parts[partIndex] = substring(\_string, startPos, stringBytes.length);

        return parts;

    }

    // Функция для поиска позиции первого вхождения подстроки в строку

function indexOf(string memory \_string, string memory \_substring) internal pure returns (uint256) {

    bytes memory stringBytes = bytes(\_string);

    bytes memory substringBytes = bytes(\_substring);

    for (uint256 i = 0; i < stringBytes.length - substringBytes.length + 1; i++) {

        bool found = true;

        for (uint256 j = 0; j < substringBytes.length; j++) {

            if (stringBytes[i + j] != substringBytes[j]) {

                found = false;

                break;

            }

        }

        if (found) {

            return i;

        }

    }

    return type(uint256).max;

}

// Функция для подсчета количества вхождений подстроки в строку

    // Функция для подсчета количества вхождений подстроки в строку

    function countOccurrences(string memory \_string, string memory \_substring) internal pure returns (uint256) {

        uint256 count = 0;

        uint256 startPos = 0;

        while (startPos < bytes(\_string).length) {

            startPos = indexOf(\_string, \_substring) + bytes(\_substring).length;

            if (startPos == type(uint256).max) {

                break;

            }

            count++;

            startPos += bytes(\_substring).length;

        }

    return count;

}

    // Функция для извлечения подстроки из строки

    function substring(string memory \_string, uint256 \_start, uint256 \_end) internal pure returns (string memory) {

        bytes memory stringBytes = bytes(\_string);

        bytes memory result = new bytes(\_end - \_start);

        for (uint256 i = \_start; i < \_end; i++) {

            result[i - \_start] = stringBytes[i];

        }

        return string(result);

    }

    // Функция для парсинга целого числа из строки

    function parseInt(string memory \_string) internal pure returns (uint256) {

        bytes memory stringBytes = bytes(\_string);

        uint256 result = 0;

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

            if (uint8(stringBytes[i]) >= 48 && uint8(stringBytes[i]) <= 57) {

                result = result \* 10 + (uint8(stringBytes[i]) - 48);

            }

        }

        return result;

    }

}

# 1.2 Функции

В данном коде были использованы следующие функции:

1. `addGradeRecord`: Функция для добавления записи об оценке в контракт.

2. `getGradeRecord`: Функция для получения записи об оценке из контракта.

3. `generateHash`: Функция для генерации уникального хеша для записи об оценке.

4. `addGradeRecordsFromXML`: Функция для добавления нескольких записей об оценках из XML.

5. `parseXMLRecord`: Функция для парсинга записи об оценке из XML.

6. `splitXMLRecords`: Функция для разделения XML на отдельные записи.

7. `getValueFromTag`: Функция для получения значения из тега XML.

8. `splitString`: Функция для разделения строки на массив по разделителю.

9. `indexOf`: Функция для поиска позиции первого вхождения подстроки в строку.

10. `countOccurrences`: Функция для подсчета количества вхождений подстроки в строку.

11. `substring`: Функция для извлечения подстроки из строки.

12. `parseInt`: Функция для парсинга целого числа из строки.

Эти функции выполняют различные задачи, связанные с добавлением и получением записей об оценках, обработкой XML-данных и обработкой строковых операций.

# 1.2 Алгоритм программы

1. Определяем структуру GradeRecord, которая содержит поля для записи оценки.

2. Создаем приватное поле gradeRecords в виде маппинга, где ключом является хеш записи, а значением - структура GradeRecord.

3. Реализуем функцию addGradeRecord для добавления записи об оценке. В функции генерируем хеш для записи, затем сохраняем запись в маппинге gradeRecords.

4. Реализуем функцию getGradeRecord для получения записи об оценке по заданным параметрам. В функции генерируем хеш для записи и получаем запись из маппинга gradeRecords.

5. Реализуем функцию generateHash для генерации уникального хеша для записи об оценке на основе студенческого номера, предмета и номера семестра.

6. Реализуем функцию addGradeRecordsFromXML для добавления нескольких записей об оценках из XML. В функции разбиваем XML на отдельные записи, затем обрабатываем каждую запись и добавляем ее в блокчейн вызывая функцию addGradeRecord.

7. Реализуем вспомогательную функцию parseXMLRecord для парсинга записи об оценке из XML. В функции получаем значения полей записи из соответствующих XML-тегов.

8. Реализуем вспомогательную функцию splitXMLRecords для разделения XML на отдельные записи. В функции разделяем XML на записи с помощью разделителя и удаляем пустые записи.

9. Реализуем вспомогательную функцию getValueFromTag для получения значения из тега XML. В функции находим начальный и конечный теги, затем извлекаем значение между ними.

10. Реализуем вспомогательную функцию splitString для разделения строки на массив по заданному разделителю.

11. Реализуем вспомогательную функцию indexOf для поиска позиции первого вхождения подстроки в строку.

12. Реализуем вспомогательную функцию countOccurrences для подсчета количества вхождений подстроки в строку.

13. Реализуем вспомогательную функцию substring для извлечения подстроки из строки.

14. Реализуем вспомогательную функцию parseInt для парсинга целого числа из строки.

В результате получается контракт GradeBook с функциональностью добавления и получения записей об оценках, а также с возможностью добавления нескольких записей сразу из XML.