Технология LINQ to XML. Учебные задания

Узлом (node) XML-документа называется любой его компонент, в частности, комментарий (comment), инструкция обработки (processing instruction), обычный текст. *Элементом* (element) XML-документа называется именованный компонент, который может содержать другие компоненты, а также иметь *атрибуты* (attributes). В объектной модели X-DOM, входящей в состав интерфейса LINQ to XML, с каждым видом компонентов XML-документа связан соответствующий класс: XNode — общий предок всех компонентов, XText — текстовый узел, т. е. узел, представляющий собой обычный текст, XComment — комментарий, XProcessingInstruction — инструкция обработки, XElement — элемент, XAttribute — атрибут, XDocument — XML-документ в целом.

Если некоторый узел *B* XML-документа содержится внутри некоторого элемента *A*, то элемент *A* называется *предком* (ancestor) узла *B*, а узел *B* — *потомком* (descendant) элемента *A*. Если элемент *A* является ближайшим предком узла *B*, то *B* называется *дочерним узлом* (child node) элемента *A*, а элемент *A* — *родительским элементом* (parent) узла *B*; если при этом узел *B* является элементом, то он называется *дочерним элементом* (child element) элемента *A*. Первый элемент XML-документа называется *корневым элементом* (root); корневой элемент является предком для всех других элементов XML-документа, сам корневой элемент предков не имеет.

Корневой элемент считается элементом *нулевого уровня*, его дочерние узлы/элементы — узлами/элементами *первого уровня*, их дочерние узлы/элементы — узлами/элементами *второго уровня*, и т. д. В XML-документе имеется единственный элемент нулевого уровня (корневой элемент), однако могут присутствовать несколько узлов нулевого уровня (комментариев или инструкций обработки).

Если в задании говорится, что элемент содержит текстовую строку (или число), то это означает, что соответствующая строка (или строковое представление числа) является дочерним текстовым узлом данного элемента.

Во всех заданиях предполагается, что элемент имеет не больше одного дочернего текстового узла, а текстовый узел содержит хотя бы один значащий символ (т. е. символ, отличный от пробела и управляющих символов). При сохранении XML-документа следует использовать метод Save класса XDocument с единственным параметром — именем файла; это обеспечит автоматическое форматирование сохраняемого документа и удаление всех текстовых узлов, не содержащих значащих символов.

Элементы, не содержащие дочерних узлов, могут представляться в двух вариантах: в виде *парных тегов*, между которыми отсутствует текст (<a>), и в виде одного *комбинированного тега* (<a />). Элементы, не содержащие узлов, могут иметь атрибуты.

Если в условии сказано, что дан XML-документ, то это означает, что дано имя файла, содержащего этот документ. Преобразование XML-документа всегда должно завершаться сохранением преобразованного документа в том же файле, из которого был считан исходный вариант этого документа.

Если в условии упоминаются порядковые номера элементов некоторой последовательности, то предполагается, что нумерация начинается от 1.

В заданиях подгрупп, предшествующих подгруппе «Работа с пространствами имен XML-документа», предполагается, что имена всех элементов и атрибутов XML-документа имеют пустое пространство имен.

Указания, приведенные к некоторым заданиям, следует учитывать и при выполнении последующих заданий текущей подгруппы.

Создание XML-документа

Во всех заданиях данной подгруппы предполагается, что исходные текстовые файлы содержат текст в кодировке «windows-1251», а все файловые строки являются непустыми. Создаваемые XML-документы также должны иметь кодировку «windows-1251».

LinqXml1. Даны имена существующего текстового файла и создаваемого XML-документа. Создать XML-документ с корневым элементом `root` и элементами первого уровня `line`, каждый из которых содержит одну строку из исходного файла.

Указание. В конструкторе корневого элемента использовать последовательность объектов XElement, полученную методом Select из исходного набора строк.

LinqXml2. Даны имена существующего текстового файла и создаваемого XML-документа. Создать XML-документ с корневым элементом `root` и элементами первого уровня, каждый из которых содержит одну строку из исходного файла и имеет имя `line` с приписанным к нему порядковым номером строки (`line1`, `line2` и т. д.).

LinqXml3. Даны имена существующего текстового файла и создаваемого XML-документа. Создать XML-документ с корневым элементом `root` и элементами первого уровня `line`, каждый из которых содержит одну строку из исходного файла. Элемент, содержащий строку с порядковым номером *N* (1, 2, …), должен иметь атрибут `num` со значением, равным *N*.

LinqXml4. Даны имена существующего текстового файла и создаваемого XML-документа. Каждая строка текстового файла содержит несколько (одно или более) слов, разделенных ровно одним пробелом. Создать XML-документ с корневым элементом `root`, элементами первого уровня `line` и элементами второго уровня `word`. Элементы `line` соответствуют строкам исходного файла и не содержат дочерних текстовых узлов, элементы `word` каждого элемента `line` содержат по одному слову из соответствующей строки (слова располагаются в алфавитном порядке).

LinqXml5. Даны имена существующего текстового файла и создаваемого XML-документа. Каждая строка текстового файла содержит несколько (одно или более) слов, разделенных ровно одним пробелом. Создать XML-документ с корневым элементом `root`, элементами первого уровня `line` и элементами второго уровня `word`. Элементы `line` соответствуют строкам исходного файла и не содержат дочерних текстовых узлов, элементы `word` каждого элемента `line` содержат по одному слову из соответствующей строки (слова располагаются в порядке их следования в исходной строке). Элемент `line` должен содержать атрибут `num`, равный порядковому номеру строки в исходном файле, элемент `word` должен содержать атрибут `num`, равный порядковому номеру слова в строке.

LinqXml6. Даны имена существующего текстового файла и создаваемого XML-документа. Каждая строка текстового файла содержит несколько (одно или более) целых чисел, разделенных ровно одним пробелом. Создать XML-документ с корневым элементом `root`, элементами первого уровня `line` и элементами второго уровня `number`. Элементы `line` соответствуют строкам исходного файла и не содержат дочерних текстовых узлов, элементы `number` каждого элемента `line` содержат по одному числу из соответствующей строки (числа располагаются в порядке убывания). Элемент `line` должен содержать атрибут `sum`, равный сумме всех чисел из соответствующей строки.

LinqXml7. Даны имена существующего текстового файла и создаваемого XML-документа. Каждая строка текстового файла содержит несколько (одно или более) целых чисел, разделенных ровно одним пробелом. Создать XML-документ с корневым элементом `root`, элементами первого уровня `line` и элементами второго уровня `sum-positive` и `number-negative`. Элементы `line` соответствуют строкам исходного файла и не содержат дочерних текстовых узлов, элемент `sum-positive` является первым дочерним элементом каждого элемента `line` и содержит сумму всех положительных чисел из соответствующей строки, элементы `number-negative` содержат по одному отрицательному числу из соответствующей строки (числа располагаются в порядке, обратном порядку их следования в исходной строке).

LinqXml8. Даны имена существующего текстового файла и создаваемого XML-документа. Каждая строка текстового файла содержит несколько (одно или более) слов, разделенных ровно одним пробелом. Создать XML-документ с корневым элементом `root`, элементами первого уровня `line`, элементами второго уровня `word` и элементами третьего уровня `char`. Элементы `line` и `word` не содержат дочерних текстовых узлов. Элементы `line` соответствуют строкам исходного файла, элементы `word` каждого элемента `line` соответствуют словам из этой строки (слова располагаются в алфавитном порядке), элементы `char` каждого элемента `word` содержат по одному символу из соответствующего слова (символы располагаются в порядке их следования в слове).

LinqXml9. Даны имена существующего текстового файла и создаваемого XML-документа. Создать XML-документ с корневым элементом `root`, элементами первого уровня `line` и комментариями (комментарии являются дочерними узлами корневого элемента). Если строка текстового файла начинается с текста «comment:», то она (без текста «comment:») добавляется в XML-документ в качестве очередного комментария, в противном случае строка добавляется в качестве дочернего текстового узла в очередной элемент `line`.

LinqXml10. Даны имена существующего текстового файла и создаваемого XML-документа. Создать XML-документ с корневым элементом `root`, элементами первого уровня `line` и инструкциями обработки (инструкции обработки являются дочерними узлами корневого элемента). Если строка текстового файла начинается с текста «data:», то она (без текста «data:») добавляется в XML-документ в качестве данных к очередной инструкции обработки с именем `instr`, в противном случае строка добавляется в качестве дочернего текстового узла в очередной элемент `line`.

Анализ содержимого XML-документа

LinqXml11. Дан XML-документ. Найти все различные имена его элементов и вывести каждое найденное имя вместе с числом его вхождений в документ. Имена элементов выводить в порядке их первого вхождения.
Указание. Использовать метод GroupBy.

LinqXml12. Дан XML-документ, содержащий хотя бы один элемент первого уровня. Найти все различные имена элементов первого уровня и вывести каждое найденное имя вместе с числом его вхождений в документ в качестве имени элемента первого уровня. Имена элементов выводить в алфавитном порядке.

LinqXml13. Дан XML-документ, содержащий хотя бы один атрибут. Вывести все различные имена атрибутов, входящих в документ. Порядок имен атрибутов должен соответствовать порядку их первого вхождения в документ.
Указание. Использовать методы SelectMany и Distinct.

LinqXml14. Дан XML-документ. Найти элементы второго уровня, имеющие дочерний текстовый узел, и вывести количество найденных элементов, а также имя каждого найденного элемента и значение его дочернего текстового узла. Порядок вывода элементов должен соответствовать порядку их следования в документе.

LinqXml15. Дан XML-документ, содержащий хотя бы один элемент первого уровня. Для каждого элемента первого уровня найти количество его потомков, имеющих не менее двух атрибутов, и вывести имя элемента первого уровня и найденное количество его потомков. Элементы выводить в алфавитном порядке их имен, а при совпадении имен — в порядке возрастания найденного количества потомков.

LinqXml16. Дан XML-документ, содержащий хотя бы один элемент первого уровня. Для каждого элемента первого уровня найти суммарное количество атрибутов у его элементов-потомков второго уровня (т. е. элементов, являющихся дочерними элементами его дочерних элементов) и вывести найденное количество атрибутов и имя элемента. Элементы выводить в порядке убывания найденного количества атрибутов, а при совпадении количества атрибутов — в алфавитном порядке имен.

LinqXml17. Дан XML-документ, содержащий хотя бы один текстовый узел. Найти все различные имена элементов, имеющих дочерний текстовый узел, и вывести эти имена, а также значения всех связанных с ними дочерних текстовых узлов. Порядок имен должен соответствовать порядку их первого вхождения в документ; текстовые значения, связанные с каждым именем, выводить в алфавитном порядке.

LinqXml18. Дан XML-документ, содержащий хотя бы один атрибут. Найти все различные имена атрибутов и вывести эти имена, а также все связанные с ними значения (все значения считаются текстовыми). Имена выводить в алфавитном порядке; значения, связанные с каждым именем, выводить в порядке их появления в документе.

LinqXml19. Дан XML-документ, содержащий хотя бы один элемент первого уровня. Для каждого элемента первого уровня найти его дочерние элементы, имеющие максимальное количество потомков (при подсчете числа потомков учитывать также потомки-узлы, не являющиеся элементами). Перебирая элементы первого уровня в порядке их появления в XML-документе, вывести имя элемента, число N — максимальное количество потомков, имеющихся у его дочерних элементов (значение N может быть равно 0), — и имена всех дочерних элементов, имеющих N потомков (имена дочерних элементов выводить в алфавитном порядке; среди этих имен могут быть совпадающие). Если элемент первого уровня не содержит дочерних элементов, то в качестве значения N выводить −1, а в качестве имени дочернего элемента — текст «no child».

LinqXml20. Дан XML-документ, содержащий хотя бы один элемент первого уровня. Для каждого элемента первого уровня найти его элементы-потомки, имеющие максимальное количество атрибутов. Перебирая элементы первого уровня в порядке их появления в XML-документе, вывести имя элемента, число N — максимальное количество атрибутов у его потомков (значение N может быть равно 0) — и имена потомков, имеющих N атрибутов (имена потомков выводить в алфавитном порядке; среди этих имен могут быть совпадающие). Если элемент первого уровня не содержит элементов-потомков, то в качестве значения N выводить −1, а в качестве имени потомка — текст «no child».

Преобразование XML-документа

LinqXml21. Дан XML-документ и строка *S*. В строке записано имя одного из некорневых элементов исходного документа. Удалить из документа все элементы первого уровня с именем *S*.

Указание. Применить метод Remove к последовательности Elements(*S*) корневого элемента.

LinqXml22. Дан XML-документ и строка *S*. В строке записано имя одного из некорневых элементов исходного документа. Удалить из документа все элементы с именем *S*.

LinqXml23. Дан XML-документ. Удалить из документа все инструкции обработки.

Указание. Для получения последовательности всех инструкций обработки воспользоваться методом OfType<XProcessingInstruction>.

LinqXml24. Дан XML-документ. Удалить из документа все комментарии, являющиеся узлами первого или второго уровня (т. е. имеющие своим родительским элементом корневой элемент или элемент первого уровня).

LinqXml25. Дан XML-документ. Для всех элементов первого и второго уровня, имеющих более одного атрибута, удалить все их атрибуты.

LinqXml26. Дан XML-документ. Для всех элементов документа удалить все их атрибуты, кроме первого.

Указание. В предикате метода Where, отбирающем все атрибуты элемента, кроме первого, использовать метод PreviousAttribute класса XAttribute.

LinqXml27. Дан XML-документ. Для всех элементов второго уровня удалить все их дочерние узлы, кроме последнего.

LinqXml28. Дан XML-документ. Удалить дочерние текстовые узлы для всех элементов третьего уровня. Если текстовый узел является единственным дочерним узлом элемента, то после его удаления элемент должен быть представлен в виде комбинированного тега.

Указание. Использовать метод OfType<XText>.

LinqXml29. Дан XML-документ. Удалить из документа все элементы первого и второго уровня, не содержащие дочерних узлов.

LinqXml30. Дан XML-документ. Удалить из документа все элементы третьего уровня, представленные комбинированным тегом.

Указание. Использовать свойство IsEmpty класса XElement.

LinqXml31. Дан XML-документ и строки S_1 и S_2. В строке S_1 записано имя одного из элементов исходного документа, строка S_2 содержит допустимое имя элемента XML. После каждого элемента первого уровня с именем S_1 добавить элемент с именем S_2. Атрибуты и потомки добавленного элемента должны совпадать с атрибутами и потомками предшествующего элемента.

Указание. Для каждого элемента S_1 вызвать метод AddAfterSelf с тремя параметрами: строкой S_2 и последовательностями Attributes и Nodes элемента S_1.

LinqXml32. Дан XML-документ и строки S_1 и S_2. В строке S_1 записано имя одного из элементов исходного документа, строка S_2 содержит допустимое имя элемента XML. Перед каждым элементом второго уровня с именем S_1 добавить элемент с именем S_2. Добавленный элемент должен содержать последний атрибут и первый дочерний элемент последующего элемента (если они есть). Если элемент S_1 не имеет дочерних элементов, то добавленный перед ним элемент S_2 должен быть представлен в виде комбинированного тега.

Указание. Использовать методы FirstOrDefault и LastOrDefault.

LinqXml33. Дан XML-документ. Для каждого элемента первого уровня, имеющего атрибуты, добавить в конец его дочерних узлов элемент с именем `attr` и атрибутами, совпадающими с атрибутами обрабатываемого элемента первого уровня, после чего удалить все атрибуты у обрабатываемого элемента. Добавленный элемент `attr` должен быть представлен в виде комбинированного тега.

Указание. Использовать метод ReplaceAttributes, указав в качестве параметра новый дочерний элемент.

LinqXml34. Дан XML-документ. Для каждого элемента первого уровня, имеющего атрибуты, добавить в конец его дочерних узлов элементы с именами, совпадающими с именами его атрибутов, и текстовыми значениями, совпадающими со значениями соответствующих атрибутов, после чего удалить все атрибуты обрабатываемого элемента первого уровня.

Указание. Использовать метод ReplaceAttributes, указав в качестве параметра последовательность элементов, полученную методом Select из последовательности атрибутов.

LinqXml35. Дан XML-документ. Для каждого элемента второго уровня добавить в конец списка его атрибутов атрибут `child-count` со значением, равным количеству всех дочерних узлов этого элемента. Если элемент не имеет дочерних узлов, то атрибут `child-count` должен иметь значение 0.

LinqXml36. Дан XML-документ. Для каждого элемента второго уровня, имеющего потомков, добавить в конец списка его атрибутов атрибут `node-count` со значением, равным количеству узлов-потомков этого элемента (всех уровней).

LinqXml37. Дан XML-документ. Для каждого элемента второго уровня, имеющего потомков, добавить к его текстовому содержимому текстовое содержимое всех элементов-потомков, после чего удалить все его узлы-потомки, кроме дочернего текстового узла.

Указание. Использовать метод SetValue и свойство Value класса XElement.

LinqXml38. Дан XML-документ. Для каждого элемента, кроме корня, изменить его имя, добавив к нему слева исходные имена всех его предков, разделенные символом «-» (дефис). Например, если корневой элемент имеет имя `root`, то элемент `bb` второго уровня, родительский элемент которого имеет имя `aa`, должен получить имя `root-aa-bb`.

Указание. Перебирая все элементы последовательности Descendants корневого элемента, использовать их свойства Parent и Name.

LinqXml39. Дан XML-документ. Для каждого элемента, кроме корня, изменить его имя, добавив к нему слева исходное имя его родительского элемента, дополненное символом «-» (дефис). Например, элемент `cc` третьего уровня, родительский элемент которого имеет имя `bb`, должен получить имя `bb-cc`.

Указание. Организовать перебор последовательности Descendants корневого элемента в обратном порядке (используя метод Reverse).

LinqXml40. Дан XML-документ. Изменить имена атрибутов всех элементов, добавив слева к исходному имени атрибута имя содержащего его элемента, дополненное символом «-» (дефис).

Указание. Так как свойство Name класса XAttribute доступно только для чтения, следует сформировать новую последовательность атрибутов с требуемыми именами и значениями (применяя метод Select к последовательности Attributes), после чего указать ее в качестве параметра метода ReplaceAttributes.

Преобразование типов при обработке XML-документа

LinqXml41. Дан XML-документ. Любой его элемент содержит либо набор дочерних элементов, либо текстовое представление вещественного числа. Добавить к каждому элементу, содержащему дочерние элементы, атрибут `sum`, равный сумме чисел, указанных в дочерних элементах. Сумма округляется до двух дробных знаков, незначащие нули не отображаются. Если ни один из дочерних элементов не содержит текстовое представление числа, то атрибут `sum` должен иметь значение 0.

Указание. Для преобразования текстового представления вещественного числа в само число достаточно выполнить приведение к типу double элемента XML, содержащего это текстовое представление. Для указания числового значения атрибута `sum` достаточно передать в качестве второго параметра конструктора XAttribute вещественное число, округленное требуемым образом (с помощью функции Math.Round с двумя параметрами).

LinqXml42. Дан XML-документ, в котором значения всех атрибутов являются текстовыми представлениями вещественных чисел. Добавить к каждому элементу первого уровня, содержащему дочерние элементы, дочерний элемент `sum`, содержащий текстовое представление суммы атрибутов всех дочерних элементов данного элемента. Сумма округляется до двух дробных знаков, незначащие нули не отображаются. Если ни один из дочерних элементов не содержит атрибутов, то элемент `sum` должен иметь значение 0.

LinqXml43. Дан XML-документ, в котором значения всех атрибутов являются текстовыми представлениями вещественных чисел. Добавить к каждому элементу первого уровня, содержащему дочерние элементы, дочерний элемент `max`, содержащий текстовое представление максимального из значений атрибутов всех элементов-потомков данного элемента. Если ни один из элементов-потомков не содержит атрибутов, то элемент `max` не добавлять.

Указание. Для единообразной обработки двух ситуаций (наличие или отсутствие атрибутов у потомков) можно построить по последовательности атрибутов последовательность числовых значений Nullable-типа double?, применить к ней метод Max и добавить новый элемент `max` с помощью метода SetElementValue, указав в качестве второго параметра результат, возвращенный методом Max. При отсутствии атрибутов у потомков метод Max вернет значение null; в этом случае метод SetElementValue не будет создавать новый элемент.

LinqXml44. Дан XML-документ и строка *S*. В строке записано имя одного из атрибутов исходного документа; известно, что все атрибуты с указанным именем содержат текстовое представление вещественного числа. Для каждого элемента выполнить следующие действия: перебирая всех его потомков, содержащих атрибут *S*, найти минимальное значение данного атрибута и записать это значение в новый атрибут `min` обрабатываемого элемента. Если ни один из потомков элемента не содержит атрибут *S*, то атрибут `min` к этому элементу не добавлять.

Указание. Использовать приведение атрибута Attribute(*S*) к Nullable-типу double?; если атрибут с указанным именем отсутствует, то будет возвращено значение null. Для создания нового атрибута с найденным минимальным значением использовать метод SetAttributeValue; в случае значения null атрибут создан не будет.

LinqXml45. Дан XML-документ. Для каждого элемента, имеющего атрибуты, добавить в начало его набора дочерних узлов элемент с именем `odd-attr-count` и логическим значением, равным true, если суммарное количество атрибутов данного элемента и всех его элементов-потомков является нечетным, и false в противном случае.

Указание. В качестве параметра конструктора XElement, определяющего значение элемента, следует использовать логическое выражение; это позволит отобразить значение логической константы в соответствии со стандартом XML.

LinqXml46. Дан XML-документ. Для каждого элемента, имеющего дочерние элементы, добавить в конец его набора атрибутов атрибут с именем `odd-node-count` и логическим значением, равным true, если суммарное количество дочерних узлов у всех его дочерних элементов является нечетным, и false в противном случае.

LinqXml47. Дан XML-документ. Для каждого элемента, имеющего хотя бы один дочерний элемент, добавить дочерний элемент с именем `has-comments` и логическим значением, равным true, если данный элемент содержит в числе своих узлов-потомков один или более комментариев, и false в противном случае. Новый элемент добавить после первого имеющегося дочернего элемента.

LinqXml48. Дан XML-документ. Для каждого элемента, имеющего не менее двух дочерних узлов, добавить дочерний элемент с именем `has-instructions` и логическим значением, равным true, если данный элемент содержит в числе своих дочерних узлов одну или более инструкций обработки, и false в противном случае. Новый элемент добавить перед последним имеющимся дочерним узлом.

LinqXml49. Дан XML-документ и строка *S*. В строке записано имя одного из атрибутов исходного документа; известно, что все атрибуты с указанным именем содержат представление некоторого промежутка времени (в днях, часах, минутах и секундах) в формате, принятом в стандарте XML. Добавить в корневой элемент атрибут `total-time`, равный суммарному значению промежутков времени, указанных во всех атрибутах *S* исходного документа.

Указание. Использовать приведение объекта Attribute(*S*) к типу TimeSpan или TimeSpan? (в зависимости от выбранного способа построения последовательности требуемых атрибутов). Для суммирования полученных отрезков времени использовать метод Aggregate и операцию «+» для типа TimeSpan.

LinqXml50. Дан XML-документ. С каждым элементом документа связывается некоторый промежуток времени (в днях, часах, минутах и секундах). Этот промежуток либо явно указывается в атрибуте `time` данного элемента (в формате, принятом в стандарте XML), либо, если данный атрибут отсутствует, считается равным одному дню. Добавить в начало набора дочерних узлов корневого элемента элемент `total-time` со значением, равным суммарному значению промежутков времени, связанных со всеми элементами первого уровня.

Указание. Использовать приведение объекта Attribute к Nullable-типу TimeSpan? и операцию ?? языка C# (бинарную операцию If языка VB.NET).

LinqXml51. Дан XML-документ. Любой его элемент содержит либо набор дочерних элементов, либо текстовое представление некоторой даты, соответствующее стандарту XML. Изменить все элементы, содержащие дату, следующим образом: добавить атрибут `year`, содержащий значение года из исходной даты, и дочерний элемент `day` с текстовым значением, равным значению дня из исходной даты, после чего удалить из обрабатываемого элемента исходную дату.

Указание. Использовать приведение элемента к типу DateTime.

LinqXml52. Дан XML-документ. С каждым элементом документа связывается некоторая дата, определяемая номерами года, месяца и дня. Компоненты этой даты указываются в атрибутах `year`, `month`, `day`. Если некоторые из этих атрибутов отсутствуют, то соответствующие компоненты определяются по умолчанию: 2000 для года, 1 для месяца и 10 для дня. Для каждого элемента добавить в начало его набора дочерних узлов элемент `date` с текстовым представлением даты, связанной с обрабатываемым элементом (дата записывается в формате, принятом в стандарте XML), и удалить имеющиеся атрибуты, связанные с компонентами даты.

Работа с пространствами имен XML-документа

LinqXml53. Дан XML-документ. В каждом элементе первого уровня определено пространство имен, распространяющееся на все его элементы-потомки. Для каждого элемента первого уровня добавить в конец его набора дочерних узлов элемент с именем `namespace` и значением, равным пространству имен обрабатываемого элемента первого уровня (пространство имен добавленного элемента должно совпадать с пространством имен его родительского элемента).

LinqXml54. Дан XML-документ, у которого корневой элемент и, возможно, какие-либо другие элементы имеют непустое пространство имен. Связать с пространством имен корневого элемента все элементы первого и второго уровня; для элементов более высоких уровней оставить их прежние пространства имен.

LinqXml55. Дан XML-документ. Преобразовать имена всех элементов второго уровня, удалив из них пространства имен (для элементов других уровней пространства имен не изменять).

LinqXml56. Дан XML-документ. В корневом элементе документа определен единственный префикс пространства имен. Снабдить этим префиксом имена всех элементов первого уровня.

LinqXml57. Дан XML-документ и строки S_1 и S_2, содержащие различные пространства имен. Определить в корневом элементе два префикса пространств имен: префикс `x`, связанный с S_1, и префикс `y`, связанный с S_2. Снабдить префиксом `x` все элементы первого уровня, а префиксом `y` — все элементы второго и последующих уровней.

LinqXml58. Дан XML-документ и строка S, содержащая некоторое пространство имен. Определить в корневом элементе префикс `node`, связанный с пространством имен, заданным в строке S, и добавить в каждый элемент первого уровня два атрибута: атрибут `node:count` со значением, равным количеству потомков-узлов для данного элемента, и атрибут `xml:count` со значением, равным количеству потомков-элементов для данного элемента (`xml` — префикс пространства имен XML).

Указание. Использовать свойство Xml класса XNamespace.

LinqXml59. Дан XML-документ. В каждом из элементов первого уровня определен единственный префикс пространства имен, причем известно, что все атрибуты с этим префиксом имеют целочисленные значения. Для каждого элемента первого уровня и его элементов-потомков удвоить значения всех атрибутов с префиксом пространства имен, определенным в этом элементе.

LinqXml60. Дан XML-документ, корневой элемент которого содержит определения двух префиксов пространств имен с именами `x` и `y`. Эти префиксы используются далее в именах некоторых элементов (у атрибутов префиксы отсутствуют). Удалить определение префикса `y` и для всех элементов, снабженных этим префиксом, заменить его на префикс `x`, а для всех элементов, снабженных префиксом `x`, заменить его на префикс `xml` пространства имен XML.

Дополнительные задания на обработку XML-документа

Во всех заданиях данной подгруппы предполагается, что в корневом элементе исходного XML-документа определено некоторое пространство имен, распространяющееся на все его элементы-потомки. Это же пространство имен необходимо использовать для имен всех элементов преобразованного документа. Префиксы пространств имен в заданиях данной подгруппы не используются.

LinqXml61. Дан XML-документ с информацией о клиентах фитнес-центра. Образец элемента первого уровня:

```
<record>
  <id>10</id>
  <date>2000-05-01T00:00:00</date>
  <time>PT5H13M</time>
</record>
```

Здесь `id` — код клиента (целое число), `date` — дата с информацией о годе и месяце, `time` — продолжительность занятий (в часах и минутах) данного клиента в течение указанного месяца. Преобразовать документ, изменив элементы первого уровня следующим образом:

```
<time id="10" year="2000" month="5">PT5H13M</time>
```

Порядок следования элементов первого уровня не изменять.

LinqXml62. Дан XML-документ с информацией о клиентах фитнес-центра. Образец элемента первого уровня (смысл данных тот же, что и в LinqXml61):

```
<time year="2000" month="5" id="10">PT5H13M</time>
```

Преобразовать документ, изменив элементы первого уровня следующим образом:

```
<id10 date="2000-05-01T00:00:00">313</id10>
```

Имя элемента должно иметь префикс `id`, после которого указывается код клиента; в значении атрибута `date` день должен быть равен 1, а время должно быть нулевым. Значение элемента равно продолжительности занятий клиента в данном месяце, переведенной в минуты. Элементы должны быть отсортированы по возрастанию кода клиента, а для одинаковых значений кода — по возрастанию даты.

LinqXml63. Дан XML-документ с информацией о клиентах фитнес-центра. Образец элемента первого уровня (смысл данных тот же, что и в LinqXml61):

```
<record date="2000-05-01T00:00:00" id="10" time="PT5H13M" />
```

Преобразовать документ, выполнив группировку данных по кодам клиентов и изменив элементы первого уровня следующим образом:

```
<client id="10">
  <time year="2000" month="5">PT5H13M</time>
  ...
</client>
```

Элементы первого уровня должны быть отсортированы по возрастанию кода клиента, их дочерние элементы — по возрастанию номера года, а для одинаковых значений года — по возрастанию номера месяца.

LinqXml64. Дан XML-документ с информацией о клиентах фитнес-центра. Образец элемента первого уровня (смысл данных тот же, что и в LinqXml61):

```
<client id="10">
  <date>2000-05-01T00:00:00</date>
  <time>PT5H13M</time>
</client>
```

Преобразовать документ, сгруппировав данные по годам, а в пределах каждого года — по месяцам. Изменить элементы первого уровня следующим образом:

```
<y2000>
  <m5>
    <client id="10" time="313" />
    ...
  </m5>
  ...
</y2000>
```

Имя элемента первого уровня должно иметь префикс y, после которого указывается номер года; имя элемента второго уровня должно иметь префикс m, после которого указывается номер месяца. Атрибут time должен содержать продолжительность занятий в минутах. Элементы первого уровня должны быть отсортированы по убыванию номера года, их дочерние элементы — по возрастанию номера месяца. Элементы третьего уровня, имеющие общего родителя, должны быть отсортированы по возрастанию кодов клиентов.

LinqXml65. Дан XML-документ с информацией о клиентах фитнес-центра. Образец элемента первого уровня (смысл данных тот же, что и в LinqXml61, данные сгруппированы по кодам клиентов; коды клиентов, снабженные префиксом id, указываются в качестве имен элементов первого уровня):

```
<id10>
  <info>
    <date>2000-05-01T00:00:00</date>
    <time>PT5H13M</time>
  </info>
  ...
</id10>
```

Преобразовать документ, сгруппировав данные по годам и изменив элементы первого уровня следующим образом:

```
<year value="2000">
  <total-time id="10">860</total-time>
  ...
</year>
```

Значение элемента второго уровня должно быть равно общей продолжительности занятий (в минутах) клиента с указанным кодом в течение указанного года. Элементы первого уровня должны быть отсортированы по возрастанию номера года, их дочерние элементы — по возрастанию кодов клиентов.

LinqXml66. Дан XML-документ с информацией о клиентах фитнес-центра. Образец элемента первого уровня (смысл данных тот же, что и в LinqXml61, данные сгруппированы по кодам клиентов):

```
<client id="10">
  <info date="2000-05-01T00:00:00" time="PT5H13M" />
  ...
</client>
```

Преобразовать документ, сгруппировав данные по годам и месяцам и оставив сведения только о тех месяцах, в которых посещали занятия не менее трех клиентов. Изменить элементы первого уровня следующим образом:

```
<d2000-5 total-time="956" client-count="3">
  <id10 time="313" />
  ...
</d2000-5>
```

Имя элемента первого уровня должно иметь префикс d, после которого указывается номер года и, через дефис, номер месяца (незначащие нули не отображаются). Имя элемента второго уровня должно иметь префикс id, после которого указывается код клиента. Атрибут total-time должен содержать суммарную продолжительность занятий (в минутах) всех клиентов в данном месяце, атрибут client-count — количество клиентов, занимавшихся в этом месяце. Атрибут time для элементов второго уровня должен содержать продолжительность занятий (в минутах) клиента с указанным кодом в данном месяце. Элементы первого уровня должны быть отсортированы по возрастанию номера года, а для одинаковых номеров года — по возрастанию номера месяца; их дочерние элементы должны быть отсортированы по возрастанию кодов клиентов.

LinqXml67. Дан XML-документ с информацией о клиентах фитнес-центра. Образец элемента первого уровня (смысл данных тот же, что и в LinqXml61):

```
<client id="10" time="PT5H13M">
  <year>2000</year>
  <month>5</month>
</client>
```

Преобразовать документ, сгруппировав данные по годам и месяцам и изменив элементы первого уровня следующим образом:

```
<y2000>
  <m1 total-time="0" client-count="0" />
  ...
  <m5 total-time="956" client-count="3" />
  ...
</y2000>
```

Имя элемента первого уровня должно иметь префикс y, после которого указывается номер года; имя элемента второго уровня должно иметь префикс m, после которого указывается номер месяца. Атрибут total-time должен содержать суммарную продолжительность занятий (в минутах) всех клиентов в данном месяце, атрибут client-count — количество клиентов, занимавшихся в этом месяце. Каждый элемент первого уровня должен содержать элементы второго уровня, соответствующие всем месяцам года; если в каком-либо месяце занятия не проводились, то атрибуты для этого месяца должны иметь нулевые значения. Элементы первого уровня должны быть отсортированы по возрастанию номера года, их дочерние элементы — по возрастанию номера месяца.

Указание. Для эффективного формирования последовательностей, связанных со всеми месяцами, использовать вспомогательную последовательность Enumerable.Range(1, 12) и метод GroupJoin.

LinqXml68. Дан XML-документ с информацией о ценах автозаправочных станций на бензин. Образец элемента первого уровня:

```
<record>
  <company>Лидер</company>
  <street>ул.Чехова</street>
  <brand>92</brand>
  <price>2200</price>
</record>
```

Здесь street — название улицы, company — название компании (названия улиц и компаний не содержат пробелов и являются допустимыми именами XML), brand — марка бензина (числа 92, 95 или 98), price — цена 1 литра бензина в копейках (целое число). Каждая компания имеет не более одной АЗС на каждой улице, цены на разных АЗС одной и той же компании могут различаться. Преобразовать документ, изменив элементы первого уровня следующим образом:

```
<station company="Лидер" street="ул.Чехова">
  <info>
    <brand>92</brand>
    <price>2200</price>
  </info>
</station>
```

Порядок следования элементов первого уровня не изменять.

LinqXml69. Дан XML-документ с информацией о ценах автозаправочных станций на бензин. Образец элемента первого уровня (смысл данных тот же, что и в LinqXml68):

```
<station company="Лидер">
  <info street="ул.Чехова">
    <brand>92</brand>
    <price>2200</price>
  </info>
</station>
```

Преобразовать документ, изменив элементы первого уровня следующим образом:

```
<b92 company="Лидер" street="ул.Чехова" price="2200" />
```

Имя элемента должно иметь префикс b, после которого указывается марка бензина. Элементы представляются комбинированными тегами и должны быть отсортированы по возрастанию марок бензина, для одинаковых марок — в алфавитном порядке названий компании, а для одинаковых компаний — в алфавитном порядке названий улиц.

LinqXml70. Дан XML-документ с информацией о ценах автозаправочных станций на бензин. Образец элемента первого уровня (смысл данных тот же, что и в LinqXml68):

```
<station brand="98" price="2850">
  <company>Лидер</company>
  <street>ул.Авиаторов</street>
</station>
```

Преобразовать документ, выполнив группировку данных по названиям компаний, а в пределах каждой компании — по маркам бензина. Изменить элементы первого уровня следующим образом:

```
<company name="Лидер">
  <brand value="98">
    <price street="ул.Авиаторов">2850</price>
    ...
  </brand>
  ...
</company>
```

Элементы первого уровня должны быть отсортированы в алфавитном порядке названий компаний, а их дочерние элементы — по убыванию марок бензина. Элементы третьего уровня, имеющие общего родителя, должны быть отсортированы в алфавитном порядке названий улиц.

LinqXml71. Дан XML-документ с информацией о ценах автозаправочных станций на бензин. Образец элемента первого уровня (смысл данных тот же, что и в LinqXml68):

```
<station street="ул.Авиаторов" company="Лидер">
  <info brand="98" price="2850" />
</station>
```

Преобразовать документ, выполнив группировку данных по маркам бензина, а в пределах каждой марки — по ценам 1 литра бензина. Изменить элементы первого уровня следующим образом:

```
<b98>
  <p2850>
    <info street="ул.Авиаторов" company="Лидер" />
    ...
  </p2850>
  ...
</b98>
```

Имя элемента первого уровня должно иметь префикс b, после которого указывается марка бензина; имя элемента второго уровня должно иметь префикс p, после которого указывается цена 1 литра бензина. Элементы первого уровня должны быть отсортированы по убыванию марок бензина, а их дочерние элементы — по убыванию цен. Элементы третьего уровня, имеющие общего родителя, должны быть отсортированы в алфавитном порядке названий улиц, а для одинаковых улиц — в алфавитном порядке названий компаний.

LinqXml72. Дан XML-документ с информацией о ценах автозаправочных станций на бензин. Образец элемента первого уровня (смысл данных тот же, что и в LinqXml68, данные сгруппированы по названиям компаний; названия компаний указываются в качестве имен элементов первого уровня):

```
<Лидер>
  <price street="ул.Чехова" brand="92">2200</price>
  ...
</Лидер>
```

Преобразовать документ, выполнив группировку данных по названиям улиц, а в пределах каждой улицы — по маркам бензина. Изменить элементы первого уровня следующим образом:

```
<ул.Чехова>
  <b92>
    <min-price company="Премьер-нефть">2050</min-price>
    ...
  </b92>
  ...
</ул.Чехова>
```

Имя элемента первого уровня совпадает с названием улицы, имя элемента второго уровня должно иметь префикс b, после которого указывается марка бензина. Значение элемента третьего уровня равно минимальной цене бензина данной марки на данной улице, его атрибут company содержит название компании, на АЗС которой предлагается минимальная цена. Элементы первого уровня должны быть отсортированы в алфавитном порядке названий улиц, а их дочерние элементы — по возрастанию марок бензина. Если имеется несколько элементов третьего уровня, имеющих общего родителя (это означает, что на одной улице имеется несколько АЗС, на которых бензин данной марки имеет минимальную цену), то эти элементы должны быть отсортированы в алфавитном порядке названий компаний.

LinqXml73. Дан XML-документ с информацией о ценах автозаправочных станций на бензин. Образец элемента первого уровня (смысл данных тот же, что и в LinqXml68, данные сгруппированы по названиям улиц; названия улиц указываются в качестве имен элементов первого уровня):

```
<ул.Чехова>
  <company name="Лидер">
    <brand>92</brand>
    <price>2200</price>
  </company>
  ...
</ул.Чехова>
```

Преобразовать документ, сгруппировав данные по названиям компаний и названиям улиц и оставив сведения только о тех АЗС, в которых предлагаются не менее двух марок бензина.

Изменить элементы первого уровня следующим образом:

```
<Лидер_ул.Чехова brand-count="2">
  <b92 price="2200" />
  <b95 price="2450" />
</Лидер_ул.Чехова>
```

Имя элемента первого уровня содержит название компании, после которого следует символ подчеркивания и название улицы; имя элемента второго уровня должно иметь префикс b, после которого указывается марка бензина. Атрибут brand-count должен содержать количество марок бензина, предлагаемых на данной АЗС. Элементы первого уровня должны быть отсортированы в алфавитном порядке названий компаний, а для одинаковых названий компаний — в алфавитном порядке названий улиц; их дочерние элементы должны быть отсортированы по возрастанию марок бензина.

LinqXml74. Дан XML-документ с информацией о ценах автозаправочных станций на бензин. Образец элемента первого уровня (смысл данных тот же, что и в LinqXml68, марки бензина, снабженные префиксом brand, указываются в качестве имен элементов первого уровня; атрибут station содержит названия улицы и компании, разделенные символом подчеркивания):

```
<brand92 station="ул.Чехова_Лидер" price="2200" />
```

Преобразовать документ, сгруппировав данные по названиям компаний и изменив элементы первого уровня следующим образом:

```
<Лидер>
  <ул.Садовая brand92="0" brand95="0" brand98="0" />
  <ул.Чехова brand92="2200" brand95="2450" brand98="0" />
  ...
</Лидер>
```

Имя элемента первого уровня совпадает с названием компании, имя элемента второго уровня совпадает с названием улицы. Атрибуты элементов второго уровня имеют префикс brand, после которого указывается марка бензина; их значением является цена 1 литра бензина указанной марки или число 0, если на данной АЗС бензин указанной марки не предлагается. Для каждой компании должна выводиться информация по каждой улице, имеющейся в исходном документе, даже если на этой улице отсутствует АЗС данной компании (в этом случае значения всех атрибутов brand должны быть равны 0). Элементы первого уровня должны быть отсортированы в алфавитном порядке названий компаний, а их дочерние элементы — в алфавитном порядке названий улиц.

LinqXml75. Дан XML-документ с информацией о ценах автозаправочных станций на бензин. Образец элемента первого уровня (смысл данных тот же, что и в LinqXml68, названия компании и улицы, разделенные символом подчеркивания, указываются в качестве имен элементов первого уровня):

```
<Лидер_ул.Чехова>
  <brand>92</brand>
  <price>2200</price>
</Лидер_ул.Чехова>
```

Преобразовать документ, сгруппировав данные по названиям улиц и изменив элементы первого уровня следующим образом:

```
<ул.Чехова>
  <brand98 station-count="0">0</brand98>
  <brand95 station-count="0">0</brand95>
  <brand92 station-count="3">2255</brand92>
</ул.Чехова>
```

Имя элемента первого уровня совпадает с названием улицы, имя элемента второго уровня имеет префикс brand, после которого указывается марка бензина. Атрибут station-count равен количеству АЗС, расположенных на данной улице и предлагающих бензин данной марки; значением элемента второго уровня является средняя цена 1 литра бензина данной марки по всем АЗС, расположенным на данной улице. Средняя цена находится по следующей формуле: «*суммарная цена по всем станциям*»/«*число станций*», где операция «/» обозначает целочисленное деление. Если на данной улице отсутствуют АЗС, предлагающие бензин данной марки, то значение соответствующего элемента второго уровня и значение его атрибута station-count должны быть равны 0. Элементы первого уровня должны быть отсортированы в алфавитном порядке названий улиц, а их дочерние элементы — по убыванию марок бензина.

LinqXml76. Дан XML-документ с информацией о задолженности по оплате коммунальных услуг. Образец элемента первого уровня:

```
<record>
  <house>12</house>
  <flat>129</flat>
  <name>Сергеев Т.М.</name>
  <debt>1833.32</debt>
</record>
```

Здесь house — номер дома (целое число), flat — номер квартиры (целое число), name — фамилия и инициалы жильца (инициалы не содержат пробелов и отделяются от фамилии одним пробелом), debt — размер задолженности в виде дробного числа: целая часть — рубли, дробная часть — копейки (незначащие нули не указываются). Все дома являются 144-квартирными, имеют 9 этажей и 4 подъезда. Преобразовать документ, изменив элементы первого уровня следующим образом:

```
<debt house="12" flat="129">
  <name>Сергеев Т.М.</name>
  <value>1833.32</value>
</debt>
```

Порядок следования элементов первого уровня не изменять.

LinqXml77. Дан XML-документ с информацией о задолженности по оплате коммунальных услуг. Образец элемента первого уровня (смысл данных тот же, что и в LinqXml76):

```
<debt house="12" flat="129" name="Сергеев Т.М.">1833.32</debt>
```

Преобразовать документ, изменив элементы первого уровня следующим образом:

```
<address12-129 name="Сергеев Т.М." debt="1833.32" />
```

Имя элемента должно иметь префикс address, после которого указывается номер дома и, через дефис, номер квартиры. Элементы представляются комбинированными тегами и должны быть отсортированы по возрастанию номеров домов, а для одинаковых номеров домов — по возрастанию номеров квартир.

LinqXml78. Дан XML-документ с информацией о задолженности по оплате коммунальных услуг. Образец элемента первого уровня (смысл данных тот же, что и в LinqXml76):

```
<debt house="12" flat="23">
  <name>Иванов А.В.</name>
  <value>1245.64</value>
</debt>
```

Преобразовать документ, выполнив группировку данных по номеру дома, а в пределах каждого дома — по номеру подъ-

езда. Изменить элементы первого уровня следующим образом:

```
<house number="12">
  <entrance number="1">
    <debt name="Иванов А.В." flat="23">1245.64</debt>
    ...
  </entrance>
  ...
</house>
```

Элементы первого уровня должны быть отсортированы по возрастанию номеров домов, а их дочерние элементы — по возрастанию номеров подъездов. Элементы третьего уровня, имеющие общего родителя, должны быть отсортированы по убыванию размера задолженности (предполагается, что размеры всех задолженностей являются различными).

LinqXml79. Дан XML-документ с информацией о задолженности по оплате коммунальных услуг. Образец элемента первого уровня (смысл данных тот же, что и в LinqXml76):

```
<house value="12">
  <flat value="129" />
  <name value="Сергеев Т.М." />
  <debt value="1833.32" />
</house>
```

Преобразовать документ, выполнив группировку данных по номеру дома, а в пределах каждого дома — по номеру этажа. Изменить элементы первого уровня следующим образом:

```
<house12>
  <floor6>
    <Сергеев_Т.М. flat="129" debt="1833.32" />
    ...
  </floor6>
  ...
</house12>
```

Имя элемента первого уровня должно иметь префикс `house`, после которого указывается номер дома; имя элемента второго уровня должно иметь префикс `floor`, после которого указывается номер этажа. Имя элемента третьего уровня совпадает с фамилией и инициалами жильца; фамилия отделяется от инициалов символом подчеркивания. Элементы первого уровня должны быть отсортированы по возрастанию номеров домов, а их дочерние элементы — по убыванию номеров этажей. Элементы третьего уровня, имеющие общего родителя, должны быть отсортированы в алфавитном порядке фамилий и инициалов жильцов.

LinqXml80. Дан XML-документ с информацией о задолженности по оплате коммунальных услуг. Образец элемента первого уровня (смысл данных тот же, что и в LinqXml76, данные сгруппированы по номерам домов; в качестве элементов второго уровня указываются фамилии и инициалы жильцов, фамилия отделяется от инициалов символом подчеркивания):

```
<house number="12">
  <Иванов_А.В.>
    <flat value="23" />
    <debt value="1245.64" />
  </Иванов_А.В.>
  ...
</house>
```

Преобразовать документ, сохранив группировку данных по номеру дома, выполнив в пределах каждого дома группировку по номеру подъезда и изменив элементы первого уровня следующим образом:

```
<house12>
  <entrance1 total-debt="2493.38" count="3">
    <flat23 name="Иванов А.В." />
    ...
  </entrance1>
  ...
</house12>
```

Имя элемента первого уровня должно иметь префикс `house`, после которого указывается номер дома, имя элемента второго уровня должно иметь префикс `entrance`, после которого указывается номер подъезда, имя элемента третьего уровня должно иметь префикс `flat`, после которого указывается номер квартиры. Атрибут `total-debt` равен суммарной задолженности жильцов данного подъезда (значение задолженности должно округляться до двух дробных знаков, незначащие нули не отображаются), атрибут `count` равен количеству задолжников в данном подъезде. Элементы первого уровня должны быть отсортированы по возрастанию номеров домов, а их дочерние элементы — по возрастанию номеров подъездов. Элементы третьего уровня, имеющие общего родителя, должны быть отсортированы по возрастанию номеров квартир. Подъезды, в которых отсутствуют задолжники, не отображаются.

LinqXml81. Дан XML-документ с информацией о задолженности по оплате коммунальных услуг. Образец элемента первого уровня (смысл данных тот же, что и в LinqXml76, данные сгруппированы по номерам домов; в качестве имен элементов первого уровня указываются номера домов, снабженные префиксом `house`, а в качестве имен элементов второго уровня — номера квартир, снабженные префиксом `flat`):

```
<house12>
  <flat23 name="Иванов А.В." debt="1245.64" />
  ...
</house12>
```

Преобразовать документ, сохранив группировку данных по номеру дома, выполнив в пределах каждого дома группировку по номеру подъезда и оставив сведения только о тех жильцах, размер задолженности которых не меньше среднего размера задолженности по данному подъезду. Изменить элементы первого уровня следующим образом:

```
<house number="12">
  <entrance number="1" count="4" avr-debt="1136">
    <debt flat="23" name="Иванов А.В.">1245.64</debt>
    <debt flat="28" name="Сидоров П.К.">1383.27</debt>
  </entrance>
  ...
</house>
```

Атрибут `count` равен количеству задолжников в данном подъезде, атрибут `avr-debt` определяет среднюю задолженность по данному подъезду в рублях (целое число), вычисленную по следующей формуле: «*суммарная задолженность в копейках*»/(«*количество задолжников*»*100) (символ «/» обозначает операцию целочисленного деления). Элементы третьего уровня содержат сведения о тех жильцах, размер задолженности которых не меньше величины `avr-debt` для данного подъезда. Элементы первого уровня должны быть отсортированы по возрастанию номеров домов, а их дочерние элементы — по возрастанию номеров подъездов. Элементы третьего уровня, имеющие общего родителя, должны быть отсортированы по возрастанию номеров квартир. Подъезды, в которых отсутствуют задолжники, не отображаются.

LinqXml82. Дан XML-документ с информацией о задолженности по оплате коммунальных услуг. Образец элемента перво-

го уровня (смысл данных тот же, что и в LinqXml76, в качестве имени элемента первого уровня указываются номера дома и квартиры, разделенные символом «-» (дефис) и снабженные префиксом addr, а в качестве значения этого элемента указывается размер задолженности для данной квартиры):

```
<addr12-23>1245.64</addr12-23>
```

Преобразовать документ, выполнив группировку данных по номеру дома, а в пределах каждого дома — по номеру этажа. Изменить элементы первого уровня следующим образом:

```
<house12>
  <floor1 count="0" total-debt="0" />
  ...
  <floor6 count="1" total-debt="1245.64" />
  ...
  <floor9 count="3" total-debt="3142.7" />
</house12>
```

Имя элемента первого уровня должно иметь префикс house, после которого указывается номер дома, имя элемента второго уровня должно иметь префикс floor, после которого указывается номер этажа. Атрибут count равен числу задолжников на данном этаже, атрибут total-debt определяет суммарную задолженность по данному этажу, округленную до двух дробных знаков (незначащие нули не отображаются). Если на данном этаже отсутствуют задолжники, то для соответствующего элемента второго уровня значения атрибутов count и total-debt должны быть равны 0. Элементы первого уровня должны быть отсортированы по возрастанию номеров домов, а их дочерние элементы — по возрастанию номеров этажей.

LinqXml83. Дан XML-документ с информацией об оценках учащихся по различным предметам. Образец элемента первого уровня:

```
<record>
  <class>9</class>
  <name>Степанова Д.Б.</name>
  <subject>Физика</subject>
  <mark>4</mark>
</record>
```

Здесь class — номер класса (целое число от 7 до 11), name — фамилия и инициалы учащегося (инициалы не содержат пробелов и отделяются от фамилии одним пробелом), subject — название предмета, не содержащее пробелов, mark — оценка (целое число в диапазоне от 2 до 5). Полных однофамильцев (с совпадающей фамилией и инициалами) среди учащихся нет. Преобразовать документ, изменив элементы первого уровня следующим образом:

```
<mark subject="Физика">
  <name class="9">Степанова Д.Б.</name>
  <value>4</value>
</mark>
```

Порядок следования элементов первого уровня не изменять.

LinqXml84. Дан XML-документ с информацией об оценках учащихся по различным предметам. Образец элемента первого уровня (смысл данных тот же, что и в LinqXml83):

```
<pupil class="9" name="Степанова Д.Б.">
  <subject>Физика</subject>
  <mark>4</mark>
</pupil>
```

Преобразовать документ, изменив элементы первого уровня следующим образом:

```
<class9 name="Степанова Д.Б." subject="Физика">4</class9>
```

Имя элемента должно иметь префикс class, после которого указывается номер класса. Элементы должны быть отсортированы по возрастанию номеров классов, для одинаковых номеров классов — в алфавитном порядке фамилий и инициалов учащихся, для каждого учащегося — в алфавитном порядке названий предметов, а для одинаковых предметов — по возрастанию оценок.

LinqXml85. Дан XML-документ с информацией об оценках учащихся по различным предметам. Образец элемента первого уровня (смысл данных тот же, что и в LinqXml83):

```
<info class="9" name="Степанова Д.Б." subject="Физика" mark="4" />
```

Преобразовать документ, выполнив группировку данных по номеру класса, в пределах каждого класса — по учащимся, а для каждого учащегося — по предметам. Изменить элементы первого уровня следующим образом:

```
<class number="9">
  <pupil name="Степанова Д.Б.">
    <subject name="Физика">
      <mark>4</mark>
      ...
    </subject>
    ...
  </pupil>
  ...
</class>
```

Элементы первого уровня должны быть отсортированы по возрастанию номеров классов, а их дочерние элементы — в алфавитном порядке фамилий и инициалов учащихся. Элементы третьего уровня, имеющие общего родителя, должны быть отсортированы в алфавитном порядке названий предметов, а элементы четвертого уровня, имеющие общего родителя, должны быть отсортированы по убыванию оценок.

LinqXml86. Дан XML-документ с информацией об оценках учащихся по различным предметам. Образец элемента первого уровня (смысл данных тот же, что и в LinqXml83):

```
<pupil name="Степанова Д.Б." class="9">
  <info mark="4" subject="Физика" />
</pupil>
```

Преобразовать документ, выполнив группировку данных по учащимся и изменив элементы первого уровня следующим образом:

```
<Степанова_Д.Б. class="9">
  <mark4 subject="Физика" />
  ...
</Степанова_Д.Б.>
```

Имя элемента первого уровня совпадает с фамилией и инициалами учащегося (пробел между фамилией и инициалами заменяется символом подчеркивания), имя элемента второго уровня должно иметь префикс mark, после которого указывается оценка. Элементы первого уровня должны быть отсортированы в алфавитном порядке фамилий и инициалов учащихся, их дочерние элементы — по убыванию оценок, а для одинаковых оценок — в алфавитном порядке названий предметов.

LinqXml87. Дан XML-документ с информацией об оценках учащихся по различным предметам. Образец элемента первого уровня (смысл данных тот же, что и в LinqXml83, данные сгруппированы по учащимся):

```
<pupil name="Степанова Д.Б." class="9">
  <mark subject="Физика">4</mark>
  ...
</pupil>
```

Преобразовать документ, выполнив группировку данных по названиям предметов и изменив элементы первого уровня следующим образом:

```
<Физика>
  <class9>
    <mark-count>4</mark-count>
    <avr-mark>4.1</avr-mark>
  </class9>
  ...
</Физика>
```

Имя элемента первого уровня совпадает с названием предмета, имя элемента второго уровня должно иметь префикс `class`, после которого указывается номер класса. Значение элемента `mark-count` равно количеству оценок по данному предмету, выставленных в данном классе; значение элемента `avr-mark` равно среднему значению этих оценок, найденному по следующий формуле: 10*«*сумма оценок*»/«*количество оценок*»*0.1 (символ «/» обозначает операцию целочисленного деления, полученное значение должно содержать не более одного дробного знака, незначащие нули не отображаются). Элементы первого уровня должны быть отсортированы в алфавитном порядке названий предметов, а их дочерние элементы — по возрастанию номеров классов.

LinqXml88. Дан XML-документ с информацией об оценках учащихся по различным предметам. Образец элемента первого уровня (смысл данных тот же, что и в LinqXml83, данные сгруппированы по классам):

```
<class number="9">
  <pupil name="Степанова Д.Б." subject="Физика" mark="4" />
  ...
</class>
```

Преобразовать документ, выполнив группировку данных по предметам и оставив сведения только о тех учащихся, которые получили по данному предмету более двух оценок. Изменить элементы первого уровня следующим образом:

```
<subject name="Физика">
  <pupil class="9" name="Степанова Д.Б." m1="4" m2="3" m3="3" />
  ...
</subject>
```

Оценки каждого учащегося по данному предмету указываются в атрибутах, имеющих префикс `m`, после которого следует порядковый номер оценки. Элементы первого уровня должны быть отсортированы в алфавитном порядке названий предметов, их дочерние элементы — по возрастанию номеров классов, а для одинаковых классов — в алфавитном порядке фамилий и инициалов учащихся. Оценки для каждого учащегося должны располагаться в порядке убывания. Если для некоторого предмета не найдены учащиеся, имеющие по нему более двух оценок, то соответствующий элемент первого уровня должен быть представлен комбинированным тегом, например:

```
<subject name="Химия" />
```

LinqXml89. Дан XML-документ с информацией об оценках учащихся по различным предметам. Образец элемента первого уровня (смысл данных тот же, что и в LinqXml83, в качестве имен элементов первого уровня указываются фамилии и инициалы учащихся; при этом пробел между фамилией и инициалами заменяется символом подчеркивания):

```
<Петров_С.Н. class="11" subject="Физика">4</Петров_С.Н.>
```

Преобразовать документ, выполнив группировку данных по предметам, а для каждого предмета — по классам. Изменить элементы первого уровня следующим образом:

```
<Физика>
  <class7 pupil-count="0" mark-count="0" />
  ...
  <class11 pupil-count="3" mark-count="5" />
</Физика>
```

Имя элемента первого уровня совпадает с названием предмета, имя элемента второго уровня должно иметь префикс `class`, после которого указывается номер класса. Значение атрибута `pupil-count` равно количеству учащихся данного класса, имеющих хотя бы одну оценку по данному предмету, значение атрибута `mark-count` равно количеству оценок по данному предмету в данном классе. Для каждого предмета должна быть выведена информация по каждому классу (от 7 до 11); если в некотором классе по данному предмету не было опрошено ни одного учащегося, то атрибуты `pupil-count` и `mark-count` должны быть равны 0. Элементы первого уровня должны быть отсортированы в алфавитном порядке названий предметов, а их дочерние элементы — по возрастанию номеров классов.

LinqXml90. Дан XML-документ с информацией об оценках учащихся по различным предметам. Образец элемента первого уровня (смысл данных тот же, что и в LinqXml84; в качестве имен элементов первого уровня указываются фамилии с инициалами учащихся и номера классов; между фамилией и инициалами указывается символ подчеркивания, а между инициалами и номером класса — дефис):

```
<Степанова_Д.Б.-9 subject="Физика" mark="4" />
```

Преобразовать документ, сгруппировав данные по номерам классов, а для каждого класса — по учащимся. Изменить элементы первого уровня следующим образом:

```
<class9>
  <Степанова_Д.Б.>
    <История count="0">0</История>
    ...
    <Физика count="3">3.3</Физика>
  </Степанова_Д.Б.>
  ...
</class9>
```

Имя элемента первого уровня должно иметь префикс `class`, после которого указывается номер класса, имя элемента второго уровня совпадает с фамилией и инициалами учащегося, между которыми указывается символ подчеркивания. Имя элемента третьего уровня совпадает с названием предмета. Значение атрибута `count` равно количеству оценок по данному предмету, полученных данным учащимся. Значение элемента третьего уровня равно средней оценке по данному предмету для данного учащегося; средняя оценка вычисляется по следующей формуле: 10*«*сумма оценок*»/«*количество оценок*»*0.1 (символ «/» обозначает операцию целочисленного деления, полученное значение должно содержать не более одного дробного знака, незначащие нули не отображаются). Для каждого учащегося должна быть выведена информация по каждому предмету, входящему в исходный XML-документ; если по некоторому предмету учащийся не имеет оценок, то значение соответствующего элемента третьего уровня и значение его атрибута `count` должны быть равны 0. Элементы первого уровня должны быть отсортированы по возрастанию номеров классов, а их дочерние элементы — в алфавитном порядке фамилий учащихся. Элементы третьего уровня, имеющие общего родителя, должны быть отсортированы в алфавитном порядке названий предметов.