Лабораторная работа № 18

Тема: Разработка программ создания и обработки XML-документов

Цель работы: Формирование умений и навыков создания и обработки XML-документов

Для чтения **XML** чаще всего используются высокоуровневые классы **XmlDocument** и **XPathDocument**. Что не удивительно, ведь ими довольно легко пользоваться. Данные представлены в логичном, понятном виде, можно в любой момент получить любой набор данных в нужно формате. Достигается это за счет того, что xml-данные загружаются в память. Это некритично для сравнительно небольших xml-файлов. Однако когда размер xml-данных доходит до нескольких сотен мегабайт, а то и гигабайт, классы **XmlDocument** или**XPathDocument** использовать просто неприлично, а в некоторых случаях и вовсе невозможно, ведь приложение будет требовать объем оперативной памяти равный размеру данных. Именно для таких случаев и предназначен класс **XmlReader**.

В отличие от своих собратьев по пространству имен, класс **XmlReader** не хранит данные в памяти. Доступ к данным осуществляется последовательно. Но из-за последовательности становится невозможным узнать, что содержит тот или иной xml-элемент, до того, как данные будут прочитаны. Этот недостаток делает класс **XmlReader** неудобным в использовании и даже может отпугнуть неопытных программистов. Но все не так страшно, как может показаться на первый взгляд.

Пример чтения XML

Чтобы было понятно, о чем идет речь, приведу простой пример. Предположим, что у нас есть xml-файл с информацией о пользователях:

Данные отдельного пользователя находятся в элементе member и содержат: идентификатор пользователя (атрибут kuid), псевдоним (элемент nickname), имя (firstName) и фамилию (lastName).

```xml
<?xml version="1.0" encoding="utf-8" ?>
<kbyte>
  <members>
    <member kuid="1">
      <nickname>Алексей</nickname>
      <firstName>Алексей</firstName>
      <lastName>Немиро</lastName>
    </member>

    <member kuid="288">
      <nickname>Игорь Голов</nickname>
      <firstName>Игорь</firstName>
      <lastName>Голов</lastName>
    </member>

    <member kuid="1858">
      <nickname>[i]Pro</nickname>
      <firstName>Артем</firstName>
      <lastName>Донцов</lastName>
    </member>

    <member kuid="2575">
      <nickname>Shark1</nickname>
      <firstName>Виталий</firstName>
      <lastName />
    </member>
  </members>
</kbyte>
```

При помощи класса **XmlDocument** данные каждого пользователя можно получить следующим образом:

```
XmlDocument xml = new XmlDocument();
xml.Load("XMLFile.xml");
foreach (XmlNode n in xml.SelectNodes("/kbyte/members/member"))
{
    Console.WriteLine("KUID: {0}", n.Attributes["kuid"].Value);
    Console.WriteLine("Псевдоним: {0}", n.SelectSingleNode("nickname").InnerText);
    Console.WriteLine("Имя: {0}", n.SelectSingleNode("firstName").InnerText);
    Console.WriteLine("Фамилия: {0}", n.SelectSingleNode("lastName").InnerText);
    Console.WriteLine("----------------------------------------");
}
```

Примечание. *Классы **XmlDocument** и **XmlNode** принадлежат пространству имен **System.Xml**. Для их использования необходимо импортировать это пространство в проект.*

Примечание. *Предполагается наличие файла с именем **XMLFile1.xml** в корневом каталоге приложения, который содержит приведенные выше xml-данные.*

При использовании класса **XPathDocument** принципиальных отличий в коде не будет.

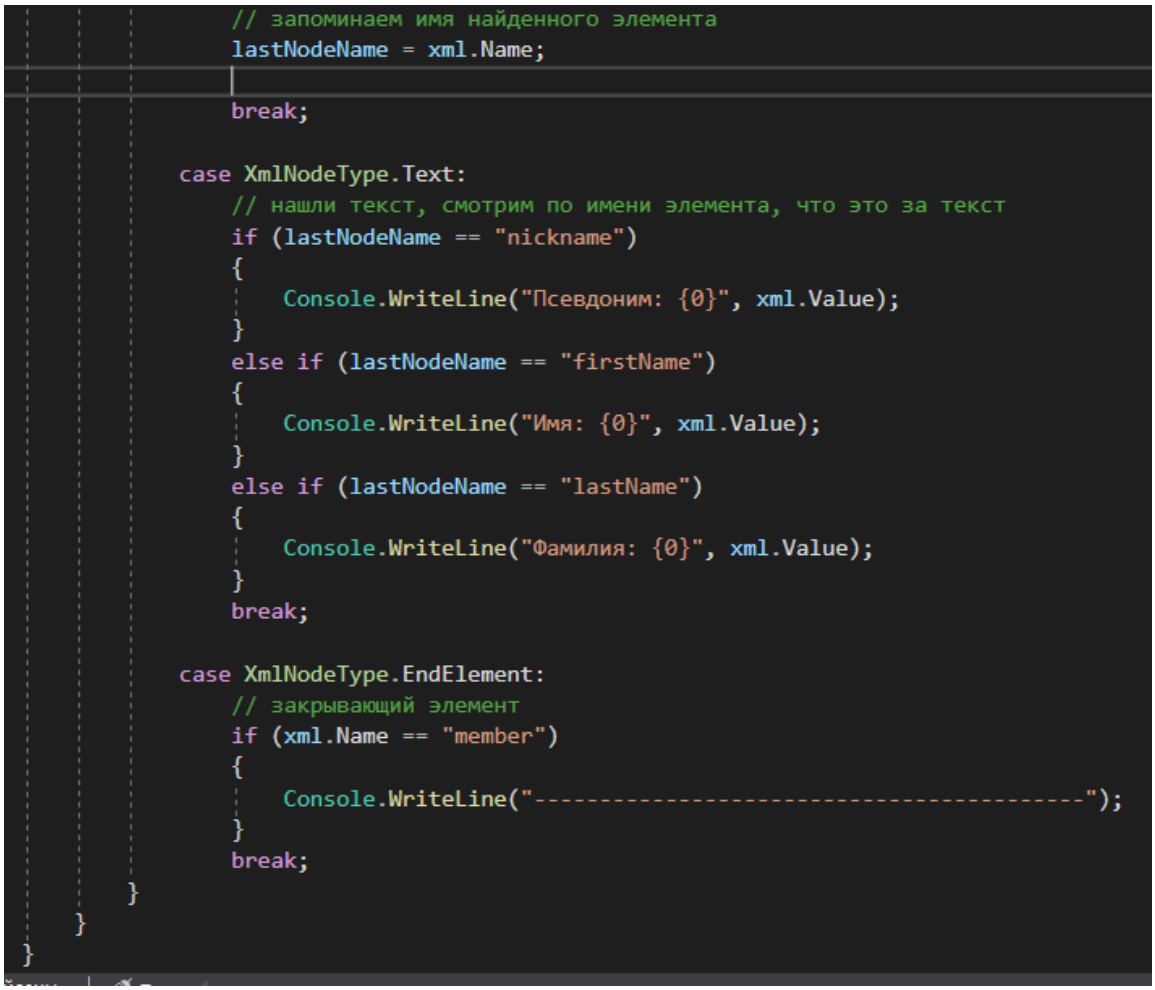
```
//Подключить namespace System.Xml.XPath;
XPathDocument xml = new XPathDocument("XMLFile1.xml");
XPathNavigator nav = xml.CreateNavigator();
foreach (XPathNavigator n in nav.Select("/kbyte/members/member"))
{
    Console.WriteLine("KUID: {0}", n.GetAttribute("kuid", ""));
    Console.WriteLine("Псевдоним: {0}", n.SelectSingleNode("nickname").Value);
    Console.WriteLine("Имя: {0}", n.SelectSingleNode("firstName").Value);
    Console.WriteLine("Фамилия: {0}", n.SelectSingleNode("lastName").Value);
    Console.WriteLine("----------------------------------------");
}
```

Примечание. *Классы **XPathDocument** и **XPathNavigator** принадлежат пространству имен **System.Xml.XPath**, которое необходимо импортировать в проект.*

Класс **XPathDocument** отличается от **XmlDocument** в основном тем, что позволяет использовать механизмы языка **XPath** (**XML Path Language**). Язык **XPath** содержит множество встроенных функций для обработки информации, но в рамках данной статьи эта тема рассматриваться не будет.

Как видино из вышеприведенных примеров, при использовании классов **XmlDocument** и **XPathDocument** доступ к данным осуществляется через коллекцию узлов (**node**), каждый из которых имеет функции и методы управления содержимым документа. В случае с классом**XmlReader** все будет менее очевидно, т.к. доступ к данным осуществляется на более низком уровне.

```
string lastNodeName = "";
using (XmlReader xml = XmlReader.Create("XMLFile1.xml"))
{
    while (xml.Read())
    {
        switch (xml.NodeType)
        {
            case XmlNodeType.Element:
                // нашли элемент member
                if (xml.Name == "member")
                {
                    if (xml.HasAttributes)
                    {
                        // поиск атрибута kuid
                        while (xml.MoveToNextAttribute())
                        {
                            if (xml.Name == "kuid")
                            {
                                Console.WriteLine("KUID: {0}", xml.Value);
                                break;
                            }
                        }
                    }
                }
```



```
                // запоминаем имя найденного элемента
                lastNodeName = xml.Name;

                break;

            case XmlNodeType.Text:
                // нашли текст, смотрим по имени элемента, что это за текст
                if (lastNodeName == "nickname")
                {
                    Console.WriteLine("Псевдоним: {0}", xml.Value);
                }
                else if (lastNodeName == "firstName")
                {
                    Console.WriteLine("Имя: {0}", xml.Value);
                }
                else if (lastNodeName == "lastName")
                {
                    Console.WriteLine("Фамилия: {0}", xml.Value);
                }
                break;

            case XmlNodeType.EndElement:
                // закрывающий элемент
                if (xml.Name == "member")
                {
                    Console.WriteLine("----------------------------------------");
                }
                break;
        }
    }
}
```

XmlReader читает данные последовательно из потока. Свойство **NodeType** содержит информацию о типе прочитанных данных. При последовательном чтении данных, невозможно сразу перейти в узел **/kbyte/members/member**. До него должен дойти указатель. Но и после того, как узел будет найден, невозможно узнать, какие данные он содержит, пока указатель не дойдет до этих данных. Код получается сложным для восприятия. И это притом, что в представленном примере структура xml-документа простая. Обработка документов со сложной структурой потребует написания более мудреного кода. Хотя, если хорошенько подумать, даже в самых тяжелых случаях можно упростить задачу. Об этом пойдет речь далее.

Работа с XmlReader

Класс **XmlReader** не имеет конструктора. Создать новый экземпляр класса **XmlReader** можно при помощи метода **Create**, который принимает либо физический путь или **url** к xml-документу, либо поток (**Stream**).

```
XmlReader xml = XmlReader.Create("XmlFile1.xml");
```

Для перехода от одного узла к другому предназначена функция **Read**, которая возвращает либо **true**, если удалось осуществить переход к узлу, либо, по достижению конца документа, **false**. Как правило, чтение всего документа производится циклом.

```
while (xml.Read())
{
// ..
}
```

В зависимости от типа узла (**NodeType**), экземпляр класса **XmlReader** может содержать дополнительно имя узла (**Name**) и его текстовое значение (**Value**).

```
Console.WriteLine($"Тип узла: {xml.NodeType}");;
Console.WriteLine($"Имя узла: {xml.Name}");
Console.WriteLine($"Значение узла: {xml.Value}" );
```

Для проверки наличия данных, предназначены вспомогательные свойства: **HasValue** – указывает, имеет узел текстовое значение (**Value**) или нет; **HasAttributes** – указывает, имеет узел атрибуты или нет; **IsEmptyElement** – содержит **true**, если элемент не имеет никаких данных.

Если текущий узел содержит атрибуты (**HasAttributes = true**), то получить их можно при помощи функции **MoveToNextAttribute**, которая, по аналогии с **Read**, будет возвращать **true**, до достижения конца определения элемента. Имя атрибута и значение также будут находиться в свойствах **Name** и **Value**.

```
if (xml.HasAttributes)
{
    while (xml.MoveToNextAttribute())
    {
        Console.WriteLine($"Атрибут: {xml.Name}");
        Console.WriteLine($"Значение: {xml.Value}");
    }
}
```

После считывания атрибутов может потребоваться вернуться назад к элементу, для этого существует метод **MoveToElement**.

```
if (xml.HasAttributes)
{
    while (xml.MoveToNextAttribute())
    {
        Console.WriteLine($"Атрибут: {xml.Name}");
        xml.MoveToElement();
        Console.WriteLine($"Значение: {xml.Value}");
    }
}
```

Для упрощения обработки документа есть несколько полезных функций. Методы **ReadOuterXml** и **ReadInnerXml** позволяют получить фрагменты **xml** в виде текста, как есть. Так, например, найдя в документе элемент **member**, при помощи функции **ReadOuterXml** можно получить его целиком и загрузить в **XmlDocument**, т.е. обработать данные на более высоком уровне, а не собирать их по крупинкам, как в продемонстрированном ранее примере.

```
if (xml.NodeType == XmlNodeType.Element)
{
    if (xml.Name == "member")
    {
        XmlDocument xmlDoc = new XmlDocument();
        xmlDoc.LoadXml(xml.ReadOuterXml());
        XmlNode n = xmlDoc.SelectSingleNode("member");
        Console.WriteLine("KUID: {0}", n.Attributes["kuid"].Value);
        Console.WriteLine("Псевдоним: {0}", n.SelectSingleNode("nickname").InnerText);
        Console.WriteLine("Имя: {0}", n.SelectSingleNode("firstName").InnerText);
        Console.WriteLine("Фамилия: {0}", n.SelectSingleNode("lastName").InnerText);
        Console.WriteLine("----------------------------------------");
    }
}
```

А при желании, можно вообще сделать класс **Member** и создать его экземпляр на основе полученной из **xml** информации при помощи механизмов десериализации**. NET Framework**.

```
[XmlRoot("member")]
Ссылок: 2
public class Member
{
    [XmlAttribute("kuid")]
    Ссылок: 0
    public int KUID { get; set; }
    [XmlElement("nickname")]
    Ссылок: 0
    public string Nickname { get; set; }
    [XmlElement("firstName")]
    Ссылок: 0
    public string FirstName { get; set; }
    [XmlElement("lastName")]
    Ссылок: 0
    public string LastName { get; set; }

    Ссылок: 0
    public Member() { }

    Ссылок: 0
    public Member(string xml)
    {
        LoadXml(xml);
    }

    ссылка: 1
    public void LoadXml(string source)
    {
        XmlSerializer mySerializer = new XmlSerializer(this.GetType());
        using (MemoryStream ms = new MemoryStream(Encoding.UTF8.GetBytes(source)))
        {
            object obj = mySerializer.Deserialize(ms);
            foreach (PropertyInfo p in obj.GetType().GetProperties())
            {
                PropertyInfo p2 = this.GetType().GetProperty(p.Name);
                if (p2 != null && p2.CanWrite)
                {
                    p2.SetValue(this, p.GetValue(obj, null), null);
                }
            }
        }
    }

}
```

Примечание. *Для использования классов xml-сериализации необходимо импортировать в проект пространство имен* ***System.Xml.Serialization****. Помимо этого, для работы функции* ***LoadXml*** *может потребоваться импортировать пространства имен* ***System.IO*** *и* ***System.Reflection****.*

```
using (XmlReader xml = XmlReader.Create("XmlFile1.xml"))
{
    while (xml.Read())
    {
        switch (xml.NodeType)
        {
            case XmlNodeType.Element:
                // нашли элемент memb
                if (xml.Name == "member")
                {
                    // передаем данные в класс Membe
                    Member m = new Member(xml.ReadOuterXml());
                    Console.WriteLine("KUID: {0}", m.KUID);
                    Console.WriteLine("Псевдоним: {0}", m.Nickname);
                    Console.WriteLine("Имя: {0}", m.FirstName);
                    Console.WriteLine("Фамилия: {0}", m.LastName);
                    Console.WriteLine("----------------------------------------");
                }
                break;
        }
    }
}
```

При этом уровень обработки данных повышается не для всего содержимого **xml**, объем которого может достигать многих гигабайт, а лишь для его небольшого фрагмента. Потребление памяти, да и в целом нагрузка на компьютер, незначительные. А вот что делать, если элемент **member** будет содержать большие объемы вложенной информации. Например, все сообщения пользователя с форумов **Kbyte.Ru**?

```xml
<?xml version="1.0" encoding="utf-8" ?>
<kbyte>
  <members>
    <member kuid="1">
      <nickname>Алексей</nickname>
      <firstName>Алексей</firstName>
      <lastName>Немиро</lastName>
      <messages>
        <message id="1">
          <subject>Привет мир!</subject>
          <text>Это тестовое сообщение.</text>
        </message>
        <message id="2">
          <subject>Как работать с XmlReader?</subject>
          <text>Это текст о том, как работать с XmlReader.</text>
        </message>
      </messages>

    </member>
    <member kuid="288">
      <nickname>Игорь Голов</nickname>
      <firstName>Игорь</firstName>
      <lastName>Голов</lastName>
    </member>

    <member kuid="1858">
      <nickname>[i]Pro</nickname>
      <firstName>Артем</firstName>
      <lastName>Донцов</lastName>
      <messages>
        <message id="3">
          <subject>Еще одно сообщение</subject>
          <text>Это текст еще одного тестового сообщения.</text>
        </message>
      </messages>
    </member>

    <member kuid="2575">
      <nickname>Shark1</nickname>
      <firstName>Виталий</firstName>
      <lastName />
      <messages />
    </member>
  </members>
</kbyte>
```

В этом документе, каждый элемент member содержит один вложенный элемент **messages**, который в свою очередь может состоять из неограниченного числа элементов **message**. Если получать все содержимое узла **member**, то потребление оперативной памяти может быть высоким.

Поэтому использование функции **ReadOuterXml** для этих целей не годится. В подобных случаях нужно разделить логику обработки элементов **member** и **messages**. Таким образом, чтобы элемент **member** содержал только основные данные о пользователе, как и в первых примерах, исключив из него элемент **messages**. Но и список сообщений никуда не пропадает, он просто будет обработан отдельно, причем ничто не запрещает использовать для этого метод **ReadOuterXml**.

Чтобы исключить объемные фрагменты **xml**, необходимо записывать в память только то, что нужно. Для этого потребуется создать экземпляр класса **XmlWriter**. Запись данных проще всего производить в **StringBuilder**, хотя можно и в поток (**Stream**). Также потребуется две дополнительных булевых (логических) переменных, одна из которых будет разрешать запись данных в экземпляр **XmlWriter**, а вторая – информировать об исключении **xml** элемента.

```
StringBuilder sb = null;
XmlWriter w = null;
XmlWriterSettings ws = new XmlWriterSettings() { Encoding =
Encoding.UTF8 };
bool isMember = false, isSkiped = false;
```

Пропустить ненужный фрагмент **xml** можно при помощи метода **Skip**.

```
using (XmlReader xml = XmlReader.Create("XmlFile1.xml"))
{
    while (xml.Read())
    {
        switch (xml.NodeType)
        {
            case XmlNodeType.Element:
                // нашли элемент member
                if (xml.Name == "member")
                {
                    isMember = true;

                    // в памяти есть данные пользователя
                    if (sb != null)
                    {
                        w.Flush();
                        w.Close();

                        // обрабатываем их
                        XmlDocument xmlDoc = new XmlDocument();
                        xmlDoc.LoadXml(sb.ToString());
                        XmlNode n = xmlDoc.SelectSingleNode("member");
                        Console.WriteLine("KUID: {0}", n.Attributes["kuid"].Value);
                        Console.WriteLine("Псевдоним: {0}", n.SelectSingleNode("nickname").InnerText);
                        Console.WriteLine("Имя: {0}", n.SelectSingleNode("firstName").InnerText);
                        Console.WriteLine("Фамилия: {0}", n.SelectSingleNode("lastName").InnerText);
                        Console.WriteLine("----------------------------------------");
                    }

                    // создаем новый XmlWirter, для записи данных пользователя
                    sb = new StringBuilder();
                    w = XmlWriter.Create(sb, ws);
                    w.WriteProcessingInstruction("xml", "version=\"1.0\" encoding=\"utf-8\"");
                }
                else if (xml.Name == "messages")
                {
                    // пропускаем фрагмент с сообщениями
                    xml.Skip();
                    // ставим отметку, что фрагмент пропущен
                    isSkiped = true;
                }
```

```
                // это данные пользователя и не пропущенный фрагмент
                if (isMember && !isSkiped)
                {
                    w.WriteStartElement(xml.Name);
                    // если есть атрибуты, записываем их
                    if (xml.HasAttributes)
                    {
                        while (xml.MoveToNextAttribute())
                        {
                            w.WriteAttributeString(xml.Name, xml.Value);
                        }
                    }
                }

                // убираем отметку о пропущенном фрагменте
                isSkiped = false;
                break;

            case XmlNodeType.Text:
                if (isMember)
                {
                    w.WriteString(xml.Value);
                }
                break;

            case XmlNodeType.EndElement:
                if (isMember)
                {
                    w.WriteFullEndElement();
                }
                if (xml.Name == "member")
                {
                    isMember = false;
                }
                break;

        }
    }
}
```

По функционалу приведенный листинг ничем не отличается от предыдущих. На выходе также будет получен фрагмент документа, содержащий только элемент **member**, исключая вложенный элемент **messages**. Что касается обработки **messages**, то для этого проще всего сделать отдельный **XmlReader** при помощи функции **ReadSubtree**.

```
else if (xml.Name == "messages")
{
    XmlReader xml2 = xml.ReadSubtree();
    while (xml2.Read())
    {
        switch (xml2.NodeType)
        {
            case XmlNodeType.Element:
                // нашли элемент message
                if (xml2.Name == "message")
                {
                    XmlDocument xmlDoc2 = new XmlDocument();
                    xmlDoc2.LoadXml(xml2.ReadOuterXml());
                    XmlNode n = xmlDoc2.SelectSingleNode("message");
                    Console.WriteLine("Сообщение # {0}", n.Attributes["id"].Value);
                    Console.WriteLine("Тема: {0}", n.SelectSingleNode("subject").InnerText);
                    Console.WriteLine("Текст: {0}", n.SelectSingleNode("text").InnerText);
                    Console.WriteLine("----------------------------------------");
                }
                break;
        }
    }

    // пропускаем фрагмент с сообщениями
    xml.Skip();
    // ставим отметку, что фрагмент пропущен
    isSkiped = true;
}
```

Таким образом, у нас будет отдельно обработан каждый элемент **member** и каждый связанный с ним **message**. Это практически никак не отразится на потреблении ресурсов, каким бы большим не был размер xml-файла.

По завершению работы с документом, всегда необходимо закрывать поток при помощи метода **Close**.

Примечание. *При использовании инструкции* ***using { }****, закрытие потока происходит автоматически. Отдельный вызов метода* ***Close*** *может привести к возникновению исключения.*

Использование сериализации для сохранения в xml.

```
using System.Xml.Serialization;
using System.IO;

namespace XMLserial
{
    [Serializable]
    ссылок 6
    public class Example
    {
        ссылка 1
        public string Name { set; get; }
        ссылка 1
        public uint Number { set; get; }
        ссылок 0
        public Example() { }
        ссылка 1
        public Example(string name,uint number)
        {
            this.Name = name;
            this.Number = number;
        }
    }
```

```
ссылок 0
static void Main(string[] args)
{
    Example[] mas = new Example[10];
    Random rnd = new Random();
    uint number = 1;
    for(int i = 0; i<mas.Length;i++)
    {
        mas[i] = new Example("Example" + i,number);
        number += (uint)rnd.Next(0, 10);
    }

    XmlSerializer formatter = new XmlSerializer(typeof(Example[]));
    //сохранить
    using(StreamWriter sw = new StreamWriter("test.xml"))
    {

        formatter.Serialize(sw, mas);
    }

    //загрузить
    using (StreamReader sr = new StreamReader("test.xml"))
    {
        Example[] newmas = (Example[])formatter.Deserialize(sr);
    }

}
```

```
//Поиск
XmlDocument xdock = new XmlDocument();
xdock.Load("test.xml");
foreach (XmlNode el in xdock.SelectNodes("//Example"))
{
    string Number = el.SelectSingleNode("Number").InnerText;
    int nub = Convert.ToInt32(Number);
    if (nub>5 && nub<20)
    {
        Console.Write("Упражнение " + el.SelectSingleNode("Name").InnerText);
        Console.WriteLine(" № {0} ", el.SelectSingleNode("Number").InnerText);
    }
}
```

Практическая часть

Задание

Разработать структуру из варианта. Сохранить и загрузить файл. Реализовать операцию из варианта

Вариант	Задание
1	**Структура** «Автосервис»: регистрационный номер автомобиля, марка, пробег, мастер, выполнивший ремонт, сумма ремонта. **Операция**: удаление каждого 2го элемента, поиск по марке автомобиля
2	**Структура** «Сотрудник»: фамилия, имя, отчество; должность; год рождения; заработная плата. **Операция**: запись нового элемента после заданного, поиск по зарплате больше заданной.
3	**Структура** «Государство»: название; столица; численность населения; занимаемая площадь. **Операция**: удаление последнего элемента, поиск по численности в границе([a,b]), где a,b – вводятся с клавиатуры.
4	**Структура** «Человек»: фамилия, имя, отчество; домашний адрес; номер телефона; возраст. **Операция**: удаление предпоследнего элемента, поиск по номеру телефона зная первые 3 цифры(из 7).
5	**Структура** «Читатель»: Фамилия И.О., номер читательского билета, название книги, срок возврата. **Операция**: запись нового элемента в середину файла, поиск по номеру читательского билета.
6	**Структура** «Школьник»: фамилия, имя, отчество; класс; номер телефона; оценки по предметам (математика, физика, русский язык, литература). **Операция**: запись нового элемента в середину файла, поиск по номеру телефона.
7	**Структура** «Студент»: фамилия, имя, отчество; домашний адрес; группа; рейтинг. **Операция**: удаление 1го элемента, поиск по фамилии.
8	**Структура** «Покупатель»: фамилия, имя, отчество; домашний адрес; номер телефона; номер кредитной карточки. **Операция**: запись нового элемента в конец, поиск по номеру кредитной карты известны 4 из 16дцати цифр.
9	**Структура** «Пациент»: фамилия, имя, отчество; домашний адрес; номер медицинской карты; номер страхового полиса. **Операция**: запись нового элемента в конец, поиск по номеру страхового полиса известны 4.
10	**Структура** «Информация»: носитель; объем; название; автор. **Операция**: удаление каждого 3го элемента, поиск по автору.
11	**Структура** «Клиент банка»: Фамилия И.О., номер счета, сумма на счете, дата последнего изменения. **Операция**: запись после заданного , поиск сумме больше заданной.
12	**Структура** «Склад»: наименование товара, цена, количество, процент торговой надбавки. **Операция**: удаление первого и последнего элемента, поиск по количеству больше заданного для произвольного поля.
13	**Структура** «Авиарейсы»: номер рейса, пункт назначения, время вылета, дата вылета, стоимость билета.

Операция: удаление каждого 2го элемента, поиск по номеру рейса.

Контрольные вопросы:

1. Особенности разметки XML документа.
2. Класс XmlNode.
3. Класс XmlNodeList.
4. Что такое XPath. И основные запросы
5. Класс XmlReader

Литература

1. Полный справочник по C#. Г. Шилдт. Издательский дом «Вильямс», 2004.
2. C# в подлиннике. Наиболее полное руководство. Х.Дейтел.
3. C# в задачах и примерах. Культин. Н.Б.
4. C# учебный курс. Г.Шилдт. СПб.: Питер, 2002.
5. C# программирование на языке высокого уровня Павловская Т.А. СПб.:

БХВ-Петербург.