Національний технічний університет України

"Київський Політехнічний Інститут"

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Курсова робота

з дисципліни "Системне програмування"

на тему

«Лексичний аналіз тексту природною мовою»

Виконав:

студент ІІІ-го курсу ФІОТ,

групи ІО-82, Міщеряков М.І.

Керівник: Пустоваров В.І.

Київ, 2010 р.

Зміст

[1 Технічне завдання 3](#_Toc281356293)

[1.1 Найменування завдання і область застосування 3](#_Toc281356294)

[1.2 Ціль та призначення 3](#_Toc281356295)

[1.3 Вимоги 3](#_Toc281356296)

[2 Вступ 4](#_Toc281356297)

[3. Обґрунтування виконання програми 5](#_Toc281356298)

[3.1 Організація лексичного аналізатора 5](#_Toc281356299)

[3.2 Особливості етапу сканування для природних мов 6](#_Toc281356300)

[3.3 Особливості етапу оцінки для природних мов 6](#_Toc281356301)

[4. Алгоритм роботи програми 8](#_Toc281356302)

[5 Опис програмних модулів 9](#_Toc281356303)

[6 Тестовий приклад 11](#_Toc281356304)

[7 Висновки 13](#_Toc281356305)

[Список використаної літератури 14](#_Toc281356306)

[Додаток А. Лістинг програми 15](#_Toc281356307)

# 1 Технічне завдання

## 1.1 Найменування завдання і область застосування

У цій роботі буде розглянуто лексичний аналіз тексту природною мовою. В якості мови розбору були вибрані мова слов’янської групи (російська). Розроблений програмний продукт виконує лексичний розбір тексту, а саме виділяє лексеми та встановлює їх тип, а також ті ознаки слів, які можна визначити на даному етапі – частина мови, рід, відмінок, число, особа та інші. Програмний продукт може бути використаний як складова частина систем перевірки правопису, управління та взаємодії з користувачами та інші.

## 1.2 Ціль та призначення

Метою розробки даного програмного продукту є закріплення умінь і навичок у програмуванні системних засобів на мові високо рівня, а також закріплення знань, отриманих при вивченні курсу "Системне програмування".

## 1.3 Вимоги

Основне завдання програми – виконати розбір вихідного тексту та отримання структури, придатної для подальшого аналізу. Одним з основних вимог є максимальна універсальність програмного продукту. Дана програма має бути розроблена на мові високого рівня, С#. Оформити результати роботи у вигляді технічної документації на проект, що включає завдання, пояснювальну записку, необхідні відомості у відповідності до ГОСТ. Надійність програмного продукту повинна бути перевірена в результаті тестування на комп'ютерах різної продуктивності, на різних версіях операційної системи Windows. Програма повинна працювати в середовищах, сумісних з ОС Windows.

# 2 Вступ

В інформатиці лексичний аналіз – процес аналітичного розбору вхідної послідовності символів(наприклад, такий як вихідний текст на одній з мов програмування) з метою отримання на виході послідовності символів, які називаються «токени». При цьому, група символів, яка ідентифікується на виході процесу як токен, називається лексема, тобто в процесі лексичного аналізу відбувається розпізнавання та виділення лексем з вихідної послідовності символів.

Як правило лексичний аналіз виконується з точки зору певної мови, чи набору мов. Мова, а точніше її граматика, задає певний набір лексем, які можуть зустрічатися на вході процесу.

Традиційно прийнято організовувати процес лексичного аналізу, розглядаючи вхідну послідовність, як потік символів. При такій організації процес самостійно керує вибором окремих символів з вхідного потоку.

Розпізнавання лексем в контексті граматики звичайно виконується шляхом їх ідентифікації (чи класифікації), згідно ідентифікаторорам (чи класам) токенів, які визначаються граматикою мови. При цьому будь-яка послідовність символів вхідного потоку (лексема), яка, відповідно граматиці, не може бути ідентифікована, як токен мови, звичайно розглядається як спеціальна токен-помилка.

Кожен токен можна представити у вигляді структури, яка містить ідентифікатор точена (чи ідентифікатор класу точена) та, якщо потрібно, послідовність символів лексеми виділеної з вхідного потоку (рядок, число і т.д.).

Ціль такої конвертації зазвичай полягає в тому, щоб підготовити вхідну послідовність для іншої програми, наприклад, для граматичного аналізатора, та звільнити його від визначення лексичних деталей в контексно-незалежній граматиці(що призвело б до ускладнення граматики).

# 3. Обґрунтування виконання програми

## 3.1 Організація лексичного аналізатора

Лексичний аналізатор(рис 3.1) – це програма чи частина програми, яка виконує лексичній аналіз. Лаксичний аналізатор звичайно працює в два етапи: сканування та оцінка.

На першій стадії, сканування, лексичний аналізатор звичайно реалізується у вигляді кінечного автомата, який визначається регулярними виразами. В ньому кодується інформація про можливі послідовності символів, які можуть зустрічатися в токенах. Наприклад, токен «ціле число» може містити будь-яку послідовність десяткових чисел. В багатьох випадках перший непробільний символ може використовуватися для визначення типу наступного токена, після чого вхідні символи обробляються один за родним, поки не зустрінеться символ, який не входить до множини допустимих символів для даного токена. В деяких мовах правила розбору лексем дещо складніші та вимагають повернень назад по послідовності, яка читається.

Отриманий таким чином токен містить необроблений вихідний текст(рядок). Для того, щоб отримати токен зі значенням, яке відповідає типу(наприклад, ціле число), виконується оцінка цього рядка – прохід по символам та обчислення значення.

Токен з типом та відповідно підготовленим значенням передається на вхід синтаксичного аналізатора.

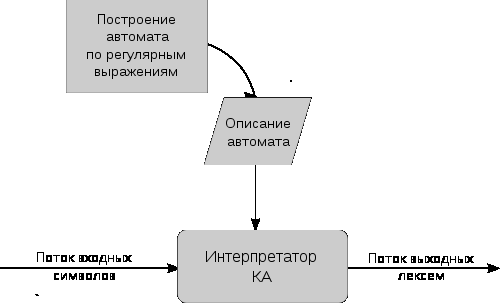


Рисунок 3.1 – Схема лексичного аналізатора

## 3.2 Особливості етапу сканування для природних мов

В більшості природніх мов можна виділити два основних токена: слова та розділові знаки. Слова містять букви та цифри(числа являються рівноправною частиною мови – числівником). Розділові знаки(в більшсті мов) – це строго визначені символи або їх послідовності, які не можуть зустрічатись в словах. В якості роздільників виступають пробіли та символ нового рядка(якщо дозволений перенос, то дефіс разом з символом нового рядка відкидаються) . Кінцем лексеми також може виступати початок розділового знаку, але потрібно не забувати, що він є лексемою.

## 3.3 Особливості етапу оцінки для природних мов

В природніх деяких природніх мовах існують строгі правила для визначення типу лексеми – частини мови, але в більшості мов просто не існує ніякого способу визначити тип лексеми, крім використання словника. В більшості мов кількість широко вживаних слів відносно невелика(<10тис. слів), що досить небагато для сучасних методів зберігання та управління даними. Але потрібно враховувати такі змінні ознаки слів, як рід, число, відмінок, час та інші, що також часто не підпорядковуються правилам. В наслідок чого словник значно розростається.

Іншою важливою особливістю є те, що лексема може мати різний тип в різних випадках і часто його можна визначити тільки на етапі синтаксичної обробки, або навіть семантичної обробки, врахувавши зміст попередніх лексем.

# 4. Алгоритм роботи програми

Програма складається з 2 частин – лексичний аналізатор та словник. В лексичному аналізаторі автомат жорстко заданий, так як для більшості мов він загалом однаковий, що дозволяє збільшити швидкодію аналізатора.

Етапи виконання основного модуля:

1. Завантаження словника.
2. Завантаження вихідного тексту в пам’ять.
3. Лексичний аналіз текту.

Словник представляє собою список виду ключ-значення, ключами являються незмінні основи слів, а значення – масив всіх допустимих варіантів змінної частини слів з іх описом – тип та ознаки(рід, число і т.д.).

Лексичній аналізатор отримує на вхід текст та словник, на виході формується масив для подальшого розбору, елемент якого має таку структуру:

class NodeStruct

{

public long X;

public int Type;

public int Index;

public string Word;

public Attributes[] Attr = new Attributes[0];

}

В ній зберігається розміщення лексеми, її тип(сума можливих типів, помилкові лексеми мають тип 0), сама лексема та всі можливі варіанти типів з атрибутами. Алгоритм роботи лексичного аналізатора приведено на рис.4.1.

В даному програмному продукті існує можливість виконувати лексичній аналіз текстів різних мов, за умови використанні відповідних словників.

На етапі отримання лексеми розділяються на 2 групи – розділові знаки та слова. Розділові знаки містять символи недопустимих для слів і можуть стояти поряд зі словами без роздільників, томі вони легко виділяються. Визначення типу виконується за допомогою словника та визначає, до якої частини мови належить слово та його ознаки такі, як рід, число та інші.



Рис. 4.1 – Алгоритм роботи лексичного аналізатора

# 5 Опис програмних модулів

До складу проекту консольної прикладної програми, орієнтованої на можливість використання в числі інших бібліотек стандартних функцій входять модулі з наступними вхідними файлами:

1. WordDictionary.cs – модуль, в якому реалізовано базовий словник. Може завантажувати записи з XML-файлу та зберігати в XML-файл.
2. AttributeModifiers.cs – модуль, в якому реалізований клас списку допустимих варіантів слова в словнику.
3. Attributes.cs – модуль, в якому реалізований базовий клас атрибутів лексеми та спеціальні класи для російської та української мов.
4. Text.cs – модуль, в якму реалізований клас для посимвольного введення вхідного тексту з файлу.
5. NodeStruct.cs – модуль, в якому реалізована базова структура елементів вихідного масиву лексем.
6. LexAn.cs – модуль, в якому реалізований клас семантичного аналізатора.
7. Program.cs – модуль, в якому реалізований клас для незалежної роботи аналізатора в консольному режимі.

# 6 Тестовий приклад

Вхідний файл для розбору (test.txt):

Привет мир!!! Я здесь.

Вхідний файл словнику (test.xml):

<**Dictionary**>

<**Word** Name="мост">

<**Forms** BaseForm="мост">

<**Form** BaseForm="мост" Name="" Type="1" Gender="geMale" Case="caNominative" Number="nuSingular"/>

<**Form** BaseForm="мост" Name="а" Type="1" Gender="geMale" Case="caGenitive" Number="nuSingular"/>

<**/Forms**>

<**/Word**>

<**Word** Name="привет">

<**Forms** BaseForm="привет">

<**Form** BaseForm="привет" Name="" Type="512"/>

<**/Forms**>

<**/Word**>

<**Word** Name="мир">

<**Forms** BaseForm="мир">

<**Form** BaseForm="мир" Name="" Type="1" Gender="geMale" Case="caNominative" Number="nuSingular"/>

<**/Forms**>

<**/Word**>

<**Word** Name="">

<**Forms** BaseForm="я">

<**Form** BaseForm="я" Name="я" Type="8" Person="peFirst" Case="caNominative" Number="nuSingular" Kind="kiProPersonal"/>

<**Form** BaseForm="я" Name="мне" Type="8" Person="peFirst" Case="caDativr" Number="nuSingular" Kind="kiProPersonal"/>

<**/Forms**>

<**/Word**>

<**Word** Name="здесь">

<**Forms** BaseForm="здесь">

<**Form** BaseForm="здесь" Name="" Type="32" Kind="kiAdvAdverbialPlace" Comparison="coNull"/>

<**/Forms**>

<**/Word**>

<**/Dictionary**>

Структура словника в пам’яті:

«привет» - незмінна частина слова

«» - Н.в, однина

«мир»

«» - іменник, Н.в., однина

«-а» – іменник, Р.в, однина

«»- незмінна частина слова

«я» – займенник, Н.в., однина

«мне» – займенник, Д.в, однина

«здесь» - незмінна частина слова

«здесь» - прислівник

Результати виконання тестового прикладу зображені на рис. 5.1.

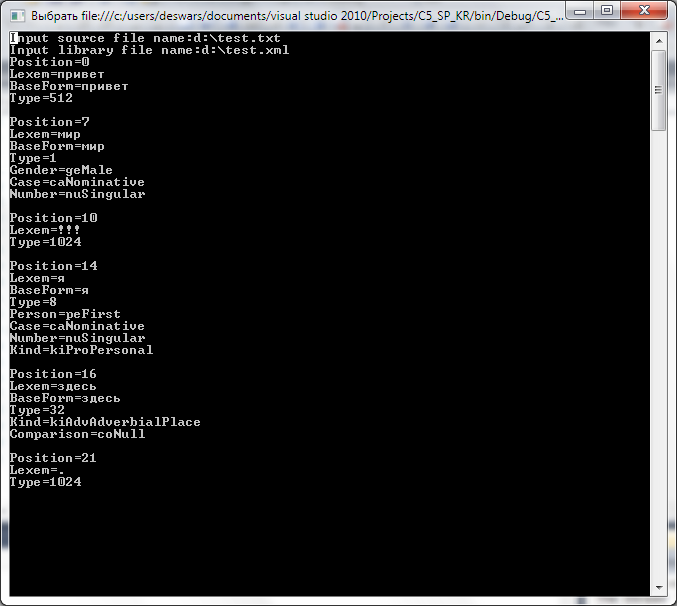


Рис. 6.1 - Результати виконання тестового прикладу

# 7 Висновки

Аналіз тексту природною мовою недзвичайно складне завдання. На кожному етапі аналізу залишаються невизначеності, які можна вирішити тільки на наступних етапах аналізу. Основна причина – не строгість задання правил або їх відсутність, виключення в правилах, неоднозначності та інше. Тому потрібно залишати додаткову інформацію та ускладнювати наступний етап аналізу, або паралельно виконувати всі етапи та повертатися при появі нової інформації, що ускладнить структуру всієї программи.

Один із альтернативних варіантів аналізу – використання шаблонів. Спочатку текст розділяється на слова, потім шукаються групи слів, які відповідають шаблонам і відразу формується результат кінцевого аналізу. Але цей метод неточний, так як втрачаються слова або значно ускладнюються шаблони та збільшується їх кількість. Після повного лексичного аналізу ми можемо використати будь-який метод для подальшого аналізу. Що являється більш універсальним.

Більшість сучасних природних мов мають подібну структуру, що робить можливим просте написання універсального лексичного аналізу для мов одного типу, на відмінну від мов програмування, які значно відрізняються в своїй структурі.

**Список використаної літератури**

1. Матеріл лекцій «Системне програмування», 5-й семестр, ФІОТ, кафедра ОТ.

2. Матеріал лабораторних робіт, розроблених Пустоваровим В.І.

3. А. Ахо, Р. Сети, Дж. Ульман "Компиляторы: принципы, технологии и инструменты", М.: "Вильямс", 2001. 768 с.

4. Steven S. Muchnik "Advanced Compiler Design And Implementation", Morgan Kaufmann Publishers, July 1997. 880 pp.

# Додаток А. Лістинг програми

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Xml;

using System.IO;

namespace C5\_SP\_KR

{

class Program

{

static void Main(string[] args)

{

Console.Write("Input source file name:");

string str = Console.ReadLine();

Text t = new Text(str);

Console.Write("Input library file name:");

str = Console.ReadLine();

WordDictionary wd = new WordDictionary(str);

LexAn la = new LexAn(t, wd);

la.Start();

la.WriteNodes();

Console.ReadLine();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

namespace C5\_SP\_KR

{

class LexAn

{

public const int spSpace = 0, spNewLine = 1, spPoint = 2, spComma = 3, spExclamation = 4,

spQuestion = 5, spColon = 6, spDash = 7, spSemicolon = 8;

Text inText;

char ch;

NodeStruct[] nodes = new NodeStruct[0];

NodeStruct ndStr;

WordDictionary dict;

char[] Separators = { ' ', '\n', '.', ',', '!', '?', ':', '-', ';' };

public LexAn(Text t, WordDictionary ind)

{

inText=t;

ch = inText.getNext();

ch = inText.getNext();

dict = ind;

}

string GetLexem()

{

string token = "";

ndStr = new NodeStruct();

ndStr.X = inText.getX() - 1;

if (ch=='\0')

return null;

bool b=false;

while (!b)

{

b = false;

for (int i = 0; i < Separators.Length; i++)

{

b |= ch == Separators[i];

}

if (!b)

{

token = token + ch;

ch = inText.getNext();

}

}

if (b && token == "")

{

b = false;

ndStr.Type = NodeStruct.tySep;

token = token + ch;

ch = inText.getNext();

if (ch=='\0')

if (token=="")

return null;

else

return token;

while (!b)

{

b = false;

for (int i = 0; i < spPoint; i++)

{

b |= ch == Separators[i];

}

if (!b)

{

token = token + ch;

ch = inText.getNext();

}

}

}

while (ch == Separators[spSpace] || ch == Separators[spNewLine])

{

ch = inText.getNext();

}

return token;

}

void DetermineType(String token)

{

KeyValuePair<String, Attributes> elem = dict.findFirst(token);

while (elem.Key != null)

{

ndStr.Type = ndStr.Type | elem.Value.getWordType();

Attributes[] tmp = (Attributes[])Array.CreateInstance(elem.Value.GetType(),ndStr.Attr.Length+1);

ndStr.Attr.CopyTo(tmp, 0);

tmp[tmp.Length - 1] = elem.Value;

ndStr.Attr = tmp;

elem = dict.findNext();

}

}

public void Start()

{

String token;

token = GetLexem();

while (token != null)

{

token = token.ToLower();

DetermineType(token);

NodeStruct[] tmp = (NodeStruct[])Array.CreateInstance(ndStr.GetType(), nodes.Length + 1);

nodes.CopyTo(tmp, 0);

tmp[tmp.Length - 1] = ndStr;

ndStr.Word = token;

nodes = tmp;

token = GetLexem();

}

}

public void WriteNodes()

{

for (int i = 0; i < nodes.Length; i++)

{

Console.WriteLine("Position="+nodes[i].X);

Console.WriteLine("Lexem=" + nodes[i].Word);

if (nodes[i].Type == NodeStruct.tySep)

{

Console.WriteLine("Type="+NodeStruct.tySep);

}

for (int k = 0; k < nodes[i].Attr.Length; k++)

{

Console.WriteLine("BaseForm=" + nodes[i].Attr[k].BaseForm);

Console.WriteLine(nodes[i].Attr[k].toString());

}

Console.WriteLine();

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.IO;

namespace C5\_SP\_KR

{

class Text

{

FileStream f;

public Text(String file)

{

string fullPath = Path.GetFullPath(file);

f = new FileStream(fullPath,FileMode.Open,FileAccess.Read);

}

public char getNext()

{

if (f.Length > f.Position)

return (char)(f.ReadByte()+(f.ReadByte()<<8));

else

return '\0';

}

public long getX()

{

return f.Position/2-1;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace C5\_SP\_KR

{

class NodeStruct

{

public const int tyNull = 0;

public const int tyNou = 1, tiNou = 0;//noun - существительное

public const int tyAdj = 2, tiAdj = 1;//adjective - прилагательное

public const int tyNum = 4, tiNum = 2;//numeral - числительное

public const int tyPro = 8, tiPro = 3;//pronoun - местоимение

public const int tyVer = 16, tiVer = 4;//verb - глагол

public const int tyAdv = 32, tiAdv = 5;//adverb - наречие

public const int tyPre = 64, tiPre = 6;//preposition - предлог

public const int tyUni = 128, tiUni = 7;//union - союз

public const int tyPar = 256, tiPar = 8;//particle - частица

public const int tyInt = 512, tiInt = 9;//interjection - междометие

public const int tySep = 1024;//separators

public long X;

public int Type;

public int Index;

public string Word;

public Attributes[] Attr = new Attributes[0];

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Xml;

namespace C5\_SP\_KR

{

class WordDictionary

{

Dictionary<String, AttributeModifiers> dict = new Dictionary<string, AttributeModifiers>();

string key,tempkey;

int size;

public WordDictionary(String file)

{

XmlTextReader xml = new XmlTextReader(file);

while (xml.Read())

{

if (xml.NodeType == XmlNodeType.Element)

{

if (xml.Name == "Word")

{

string key = xml.GetAttribute("Name");

AttributeModifiers attrm = new AttributeModifiers();

attrm.Load(xml);

dict.Add(key, attrm);

}

}

}

}

public WordDictionary()

{

}

public void Add(string word, AttributeModifiers attrm)

{

dict.Add(word, attrm);

}

public void Save(String file)

{

XmlTextWriter xml = new XmlTextWriter(file, System.Text.Encoding.Unicode);

xml.WriteStartDocument();

xml.WriteStartElement("Dictionary");

foreach (KeyValuePair<String,AttributeModifiers> el in dict)

{

xml.WriteStartElement("Word");

xml.WriteAttributeString("Name", el.Key);

el.Value.Save(xml);

xml.WriteEndElement();

}

xml.WriteEndElement();

xml.WriteEndDocument();

xml.Close();

}

public KeyValuePair<String, Attributes> findFirst(String token)

{

key = token;

size = 0;

tempkey = "";

KeyValuePair<String, Attributes> res;

AttributeModifiers attrm;

if (dict.TryGetValue(tempkey, out attrm))

{

Attributes attr = attrm.find(key);

if (attr != null)

{

res = new KeyValuePair<string, Attributes>(attrm.BaseForm, attr);

return res;

}

}

return findNext();

}

public KeyValuePair<String, Attributes> findNext()

{

size++;

while (size <= key.Length)

{

KeyValuePair<String, Attributes> res;

AttributeModifiers attrm;

tempkey = key.Substring(0, size);

if (dict.TryGetValue(tempkey, out attrm))

{

Attributes attr = attrm.find(key.Substring(size));

if (attr != null)

{

res = new KeyValuePair<string, Attributes>(attrm.BaseForm, attr);

return res;

}

}

size++;

}

return new KeyValuePair<string,Attributes>();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Xml;

namespace C5\_SP\_KR

{

class AttributeModifiers

{

Dictionary<String, Attributes> table = new Dictionary<String, Attributes>();

public string BaseForm;

public Attributes find(String token)

{

Attributes attr;

bool b = table.TryGetValue(token, out attr);

if (b)

return attr;

return null;

}

public void Add(String key, Attributes value)

{

table.Add(key, value);

}

public void Save(XmlTextWriter xml)

{

xml.WriteStartElement("Forms");

xml.WriteAttributeString("BaseForm", BaseForm);

foreach (KeyValuePair<String, Attributes> el in table)

{

xml.WriteStartElement("Form");

xml.WriteAttributeString("Name", el.Key);

el.Value.Save(xml);

xml.WriteEndElement();

}

xml.WriteEndElement();

}

public void Load(XmlTextReader xml)

{

xml.Read();

if (xml.NodeType == XmlNodeType.Element)

{

if (xml.Name == "Forms")

{

BaseForm = xml.GetAttribute("BaseForm");

while (xml.Read() && xml.NodeType == XmlNodeType.Element)

{

if (xml.Name == "Form")

{

string key = xml.GetAttribute("Name");

Attributes attr = new Attributes();

if (xml.NodeType == XmlNodeType.Element)

{

string type = xml.GetAttribute("Type");

if (type == NodeStruct.tyNou.ToString())

{

attr = new AttributesNoun();

attr.Load(xml);

}

if (type == NodeStruct.tyAdj.ToString())

{

attr = new AttributesAdjective();

attr.Load(xml);

}

if (type == NodeStruct.tyNum.ToString())

{

attr = new AttributesNumeral();

attr.Load(xml);

}

if (type == NodeStruct.tyPro.ToString())

{

attr = new AttributesPronoun();

attr.Load(xml);

}

if (type == NodeStruct.tyVer.ToString())

{

attr = new AttributesVerb();

attr.Load(xml);

}

if (type == NodeStruct.tyAdv.ToString())

{

attr = new AttributesAdverb();

attr.Load(xml);

}

if (type == NodeStruct.tyPre.ToString())

{

attr = new AttributesPreposition();

attr.Load(xml);

}

if (type == NodeStruct.tyUni.ToString())

{

attr = new AttributesUnion();

attr.Load(xml);

}

if (type == NodeStruct.tyPar.ToString())

{

attr = new AttributesParticle();

attr.Load(xml);

}

if (type == NodeStruct.tyInt.ToString())

{

attr = new AttributesInterjection();

attr.Load(xml);

}

}

table.Add(key, attr);

}

}

}

}

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Xml;

namespace C5\_SP\_KR

{

enum eGender { geNull=0 ,geMale, geFemale, geDual, geCommon, geInaimate };

enum eCase { caNull=0, caNominative, caGenitive, caDativr, caAccusative, caAblative, caPreposition };

enum eNumber { nuNull=0, nuSingular, nuPlural };

enum eComparation { coNull=0, coPositive, coComparative, coSuperlative };

enum ePerson { peNull=0, peFirst, peSecond, peThird };

enum eInclination { inNull=0, inRealist, inImperative, inConditional };

enum eTime { tiNull=0, tiPresent, tiPast, tiFuture };

enum eForm { foNull=0, foImperfective, foPerfective };

enum eKindAdjective { kiNull=0, kiAdjQualitive, kaAdjRelative, kaAdjPossessive };

enum eKindNumeral { kiNull=0, kiNumCardinal, kiNumOrdinal, kiNumPartitive, kiNumMultiplicative, kiNumCollective, kiNumDistributive };

enum eKindPronoun { kiNull=0, kiProPersonal, kiProReflexive, kiProPossessivve, kiProInterrogative, kiProIndication, kiProDistributive, kiProNegative, kiProMutual };

enum eKindAdverb { kiNull=0, kiAdvAdverbialPlace, kiAdvAdverbialTime, kiAdvAdverbialReason, kiAdvAdverbialGoal, kiAdvIdentificationQualitative, kiAdvIdentificationQuantitative, kiAdvIdentificationMethod, kiAdvIdentificationComparison, kiAdvIdentificationAggregate };

enum eKindPreposition { kiNull=0, kiPreSpatial, kiPreTemporal, kiPreComparison, kiPreContent, kiPreAgent, kiPreInstrument, kiPreCause, kiPreReference };

enum eKindUnion { kiNull=0, kiUniConnecting, kiUniDividing, kiUniComparative, kiUniComparative1, kiUniExplenatory, kiUniExplanatory, kiUniTemporary, kiUniContingent, kiUniCasual, kiUniConcessive, kiUniRestrictive, kiUniGradation };

class Attributes

{

public const int type = NodeStruct.tyNull;

public string BaseForm;

public virtual void Save(XmlWriter xml)

{

xml.WriteAttributeString("Type", type.ToString());

}

public virtual void Load(XmlTextReader xml)

{

string str = xml.GetAttribute("BaseForm");

BaseForm = str;

}

public virtual int getWordType()

{

return type;

}

public virtual string toString()

{

return type+"";

}

}

class AttributesNoun : Attributes

{

public const int type = NodeStruct.tyNou;

public eGender Gender;//род

public eCase Case;//падеж

public eNumber Number;//число

public override void Save(XmlWriter xml)

{

xml.WriteAttributeString("BaseForm", BaseForm);

xml.WriteAttributeString("Type", type.ToString());

xml.WriteAttributeString("Gender", Gender.ToString());

xml.WriteAttributeString("Case", Case.ToString());

xml.WriteAttributeString("Number", Number.ToString());

}

public override void Load(XmlTextReader xml)

{

base.Load(xml);

string str;

str = xml.GetAttribute("Gender");

Gender = (eGender)eGender.Parse(typeof(eGender), str);

str = xml.GetAttribute("Case");

Case = (eCase)eCase.Parse(typeof(eCase), str);

str = xml.GetAttribute("Number");

Number = (eNumber)eNumber.Parse(typeof(eNumber), str);

}

public virtual int getWordType()

{

return type;

}

public override string toString()

{

return "Type="+type+"\nGender="+Gender + "\nCase=" + Case + "\nNumber=" + Number;

}

}

class AttributesAdjective : Attributes

{

public const int type = NodeStruct.tyAdj;

public eGender Gender;//род

public eCase Case;//падеж

public eNumber Number;//число

public eKindAdjective Kind;//разряд

public eComparation Comparison;//степень сравнения

public override void Save(XmlWriter xml)

{

xml.WriteAttributeString("BaseForm", BaseForm);

xml.WriteAttributeString("Type", type.ToString());

xml.WriteAttributeString("Gender", Gender.ToString());

xml.WriteAttributeString("Case", Case.ToString());

xml.WriteAttributeString("Number", Number.ToString());

xml.WriteAttributeString("Kind", Kind.ToString());

xml.WriteAttributeString("Comparison", Comparison.ToString());

}

public override void Load(XmlTextReader xml)

{

base.Load(xml);

string str;

str = xml.GetAttribute("Gender");

Gender = (eGender)eGender.Parse(typeof(eGender), str);

str = xml.GetAttribute("Case");

Case = (eCase)eCase.Parse(typeof(eCase), str);

str = xml.GetAttribute("Number");

Number = (eNumber)eNumber.Parse(typeof(eNumber), str);

str = xml.GetAttribute("Kind");

Kind = (eKindAdjective)eKindAdjective.Parse(typeof(eKindAdjective), str);

str = xml.GetAttribute("Comparison");

Comparison = (eComparation)eComparation.Parse(typeof(eComparation), str);

}

public override string toString()

{

return "Type=" + type + "\nGender=" + Gender + "\nCase=" + Case + "\nNumber=" + Number + "\nKind=" + Kind + "\nComparison=" + Comparison;

}

public virtual int getWordType()

{

return type;

}

}

class AttributesNumeral : Attributes

{

public const int type = NodeStruct.tyNum;

public eGender Gender;//род

public eCase Case;//падеж

public eKindNumeral Kind;//вид

public override void Save(XmlWriter xml)

{

xml.WriteAttributeString("BaseForm", BaseForm);

xml.WriteAttributeString("Type", type.ToString());

xml.WriteAttributeString("Gender", Gender.ToString());

xml.WriteAttributeString("Case", Case.ToString());

xml.WriteAttributeString("Kind", Kind.ToString());

}

public override void Load(XmlTextReader xml)

{

base.Load(xml);

string str;

str = xml.GetAttribute("Gender");

Gender = (eGender)eGender.Parse(typeof(eGender), str);

str = xml.GetAttribute("Case");

Case = (eCase)eCase.Parse(typeof(eCase), str);

str = xml.GetAttribute("Kind");

Kind = (eKindNumeral)eKindNumeral.Parse(typeof(eKindNumeral), str);

}

public override string toString()

{

return "Type=" + type + "\nGender=" + Gender + "\nCase=" + Case + "\nKind=" + Kind;

}

public virtual int getWordType()

{

return type;

}

}

class AttributesPronoun : Attributes

{

public const int type = NodeStruct.tyPro;

public ePerson Person;//лицо

public eCase Case;//падеж

public eNumber Number;//число

public eKindPronoun Kind;//вид

public override void Save(XmlWriter xml)

{

xml.WriteAttributeString("BaseForm", BaseForm);

xml.WriteAttributeString("Type", type.ToString());

xml.WriteAttributeString("Person", Person.ToString());

xml.WriteAttributeString("Case", Case.ToString());

xml.WriteAttributeString("Number", Number.ToString());

xml.WriteAttributeString("Kind", Kind.ToString());

}

public override void Load(XmlTextReader xml)

{

base.Load(xml);

string str;

str = xml.GetAttribute("Person");

Person = (ePerson)ePerson.Parse(typeof(ePerson), str);

str = xml.GetAttribute("Case");

Case = (eCase)eCase.Parse(typeof(eCase), str);

str = xml.GetAttribute("Number");

Number = (eNumber)eNumber.Parse(typeof(eNumber), str);

str = xml.GetAttribute("Kind");

Kind = (eKindPronoun)eKindPronoun.Parse(typeof(eKindPronoun), str);

}

public override string toString()

{

return "Type=" + type + "\nPerson=" + Person + "\nCase=" + Case + "\nNumber=" + Number + "\nKind=" + Kind;

}

public virtual int getWordType()

{

return type;

}

}

class AttributesVerb : Attributes

{

public const int type = NodeStruct.tyVer;

public eInclination Inclination;//наклонение

public eTime Time;//время

public eGender Gender;//род

public ePerson Person;//лицо

public eNumber Number;//число

public eForm Form;//форма

public override void Save(XmlWriter xml)

{

xml.WriteAttributeString("BaseForm", BaseForm);

xml.WriteAttributeString("Type", type.ToString());

xml.WriteAttributeString("Incination", Inclination.ToString());

xml.WriteAttributeString("Time", Time.ToString());

xml.WriteAttributeString("Gender", Gender.ToString());

xml.WriteAttributeString("Person", Person.ToString());

xml.WriteAttributeString("Numder", Number.ToString());

xml.WriteAttributeString("Form", Form.ToString());

}

public override void Load(XmlTextReader xml)

{

base.Load(xml);

string str;

str = xml.GetAttribute("Inclination");

Inclination = (eInclination)eInclination.Parse(typeof(eInclination), str);

str = xml.GetAttribute("Time");

Time = (eTime)eTime.Parse(typeof(eTime), str);

str = xml.GetAttribute("Gender");

Gender = (eGender)eGender.Parse(typeof(eGender), str);

str = xml.GetAttribute("Person");

Person = (ePerson)ePerson.Parse(typeof(ePerson), str);

str = xml.GetAttribute("Number");

Number = (eNumber)eNumber.Parse(typeof(eNumber), str);

str = xml.GetAttribute("Form");

Form = (eForm)eForm.Parse(typeof(eForm), str);

}

public override string toString()

{

return "Type=" + type + "\nGender=" + Gender + "\nPerson=" + Person + "\nNumber=" + Number + "\nForm=" + Form;

}

public virtual int getWordType()

{

return type;

}

}

class AttributesAdverb : Attributes

{

public const int type = NodeStruct.tyAdv;

public eKindAdverb Kind;//вид

public eComparation Comparison;//степень сравнения

public override void Save(XmlWriter xml)

{

xml.WriteAttributeString("BaseForm", BaseForm);

xml.WriteAttributeString("Type", type.ToString());

xml.WriteAttributeString("Kind", Kind.ToString());

xml.WriteAttributeString("Comparison", Comparison.ToString());

}

public override void Load(XmlTextReader xml)

{

base.Load(xml);

string str;

str = xml.GetAttribute("Kind");

Kind = (eKindAdverb)eKindAdverb.Parse(typeof(eKindAdverb), str);

str = xml.GetAttribute("Comparison");

Comparison = (eComparation)eComparation.Parse(typeof(eComparation), str);

}

public override string toString()

{

return "Type=" + type + "\nKind=" + Kind + "\nComparison=" + Comparison;

}

public virtual int getWordType()

{

return type;

}

}

class AttributesPreposition : Attributes

{

public const int type = NodeStruct.tyPre;

public eKindPreposition Kind;//вид

public override void Save(XmlWriter xml)

{

xml.WriteAttributeString("BaseForm", BaseForm);

xml.WriteAttributeString("Type", type.ToString());

xml.WriteAttributeString("Kind", Kind.ToString());

}

public override void Load(XmlTextReader xml)

{

base.Load(xml);

string str;

str = xml.GetAttribute("Kind");

Kind = (eKindPreposition)eKindPreposition.Parse(typeof(eKindPreposition), str);

}

public override string toString()

{

return "Type=" + type + "\nKind=" + Kind;

}

public virtual int getWordType()

{

return type;

}

}

class AttributesUnion : Attributes

{

public const int type = NodeStruct.tyUni;

public eKindUnion Kind;//вид

public override void Save(XmlWriter xml)

{

xml.WriteAttributeString("BaseForm", BaseForm);

xml.WriteAttributeString("Type", type.ToString());

xml.WriteAttributeString("Kind", Kind.ToString());

}

public override void Load(XmlTextReader xml)

{

base.Load(xml);

string str;

str = xml.GetAttribute("Kind");

Kind = (eKindUnion)eKindUnion.Parse(typeof(eKindUnion), str);

}

public override string toString()

{

return "Type=" + type + "\nKind=" + Kind;

}

public virtual int getWordType()

{

return type;

}

}

class AttributesParticle : Attributes

{

public const int type = NodeStruct.tyPar;

public override string toString()

{

return "Type=" + type;

}

public virtual int getWordType()

{

return type;

}

public override void Save(XmlWriter xml)

{

xml.WriteAttributeString("BaseForm", BaseForm);

xml.WriteAttributeString("Type", type.ToString());

}

}

class AttributesInterjection : Attributes

{

public const int type = NodeStruct.tyInt;

public override string toString()

{

return "Type=" + type;

}

public virtual int getWordType()

{

return type;

}

public override void Save(XmlWriter xml)

{

xml.WriteAttributeString("BaseForm", BaseForm);

xml.WriteAttributeString("Type", type.ToString());

}

}

}