МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«Київський політехнічний інститут»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра СКС

Лабораторна робота №5

з дисципліни

системне програмне забезпечення

на тему:

**«Синтаксичний аналізатор (парсер)»**

Виконав студент Перевірив:

Групи КВ-64 \_\_\_\_\_\_\_\_\_\_\_

### Подольський Сергій Валентинович

залікова книжка № КВ6415 \_\_\_\_\_\_\_\_\_\_\_\_\_(бали)

#### VІІ семестр

Київ-2009

1. **Краткая теория и постановка задачи**

Задача синтаксического анализатора (парсера) заключается в проверки соответ-ствия входного файла допустимому формату и в выявлении из символов входного файла синтаксических конструкций. Входом для парсера являются поток символов и определение формата синтаксиса. Выходом для парсера являются распознанные синтаксические конструкции.

Парсер можно построить несколькими способами. Один из способов – это разбор входного файла при помощи автомата. Символ из входного файла и текущее со-стояние автомата определяют действие и следующее состояние автомата. Этот процесс можно отобразить следующим образом:

Автомат(символ, состояние) → (действие, состояние)

Преимущество такого метода построения парсера в том, что скорость разбора входного файла не зависит от сложности проверяемого формата. Логика работы автомата одинакова как для простых, так и для сложных форматов. Применение автоматов позволяет создавать унифицированный интерфейс для разбора файлов.

Автомат можно представить таблицей. Координата столбца таблицы соответствует номеру состояния автомата, а координата строки таблицы соответствует коду символа. Нет необходимости создавать строки для каждого возможного символа, так как некоторые символы требуют одинаковой обработки. Для этого каждому символу назначается класс символа (некоторое число). Метод перекодирования символов в классы символов зависит от количества классов.

Если синтаксис входного файла приводит к созданию большого автомата, то вместо одного автомата можно построить дерево автоматов. Сначала создаются и отлаживаются небольшие автоматы для разбора некоторых частей синтаксиса. Затем создаётся главный автомат, который связывает в дерево все автоматы. Использование дерева автоматов позволяет использовать уже готовые автоматы при построении новых.

## Задание на работу

Разработать инфраструктуру (структуры данных, константы, набор функций, методов, классов, порядок работы и т. п.) для построения парсера на основе работы автоматов.

Необходимые условия:

1. Символы входного файла могут быть в любой кодировке (один символ занимает не обязательно 1 байт), входной файл состоит из любого количества символов;
2. Один парсер может состоять из нескольких автоматов, всегда есть один главный автомат, из главного автомата можно переходить в другие автоматы парсера и т. д. вниз по дереву;
3. Каждый автомат управляется одним входным символом, текущим состоянием автомата и возможно текущим контекстом разбора;
4. Должна быть функция/метод получения очередной синтаксической конструкции из входного файла или получение информации об ошибке разбора.

Используя разработанную инфраструктуру создать парсеры для следующих входных файлов:

* 1. Входной файл состоит из email адресов разделённых пробелами или символами перевода строки, синтаксическая конструкция – это имя и домен. Email адрес состоит из имени, символа ‘@’ и доменного имени. Имя состоит из букв английского алфавита, цифр и символа подчёркивания. Доменное имя состоит из одного или нескольких имён поддоменов, разделённых символом ‘.’ (точка). Имя поддомена состоит из одной или нескольких букв английского алфавита, цифр и символа ‘-‘ (минус), должно начинаться и заканчиваться буквой или цифрой.
  2. Входной файл имеет следующий синтаксис:

1. Входной файл состоит из комментариев, секций и параметров, используются ASCII символы.
2. Комментарий может начинаться с символа ‘#’ и заканчиваться символом новой строки или концом файла.
3. Комментарий может начинаться с последовательности символов /\* и заканчиваться последовательностью символов \*/.
4. Комментарии могут быть расположены в любом месте входного файла.
5. Параметр начинается с имени параметра, далее идёт символ '=', затем значение параметра, значение параметра завершается символом ';'.
6. Секция начинается с имени секции, далее идёт опциональное значение, которое завершается символом '{', секция завершается символом '}'. Внутри секции могут быть комментарии, параметры и секции.
7. Значения могут состоять из любых символов, разделенных пробелами и символами новой строки.
8. Значения могут содержать строки, заключённые в двойные кавычки. Двойная кавычка в строке кодируется двумя символами \", символ '\' кодируется – \\, символ новой строки кодируется – \n и символ табуляции кодируется – \t. Любой другой символ после '\' считается ошибкой.
9. Справа и слева от '=' после имени параметра, справа и слева от ‘;’ в конце значения параметра, справа и слева от '{' и '}' в секции и между именем секции и опциональным значением допускается любое количество пробелов и символов новой строки.
10. Формат значений не оговаривается. Имя параметра или секции должно начинаться с буквы, далее могут идти буквы, цифры и символ подчеркивания.

Разбор синтаксиса в пунктах b, c, h необходимо реализовать в отдельных автоматах.

Синтаксическая конструкция – это тип конструкции (начало секции, конец секции, параметр) и, в зависимости от типа конструкции, имя секции плюс опциональное значение, ничего, имя параметра плюс значение.

Если в значении присутствуют несколько пробелов, символов табуляции или символов перевода строки, то их необходимо заменить одиночным пробелом.

**Ограничения при реализации**

При выполнении этой работы нельзя использовать генераторы лексических и синтаксических анализаторов, регулярные выражения и т. п.

## Отчёт

Отчёт должен содержать:

1. Описание идеи разработанной инфраструктуры.
2. Графическое представление разработанной инфраструктуры.
3. Графическое представление автоматов для каждого синтаксиса.
4. Листинг разработанной инфраструктуры.
5. Листинг двух парсеров.
6. Пример работы парсера для каждого синтаксиса.
7. **Описание разработанного алгоритма.**

Приложение осуществляет разбор синтаксиса, тип которого указывается пользователем.

Для первого синтаксиса (список адресов электронной почты) создан один автомат, для второго – дерево автоматов. Каждый автомат содержит две таблицы:

* 1. таблица состояний, каждой ячейкой которой является состояние автомата;
  2. таблица действий, каждой ячейкой которой является метод по обработке переключения автомата в новое состояние.

Таким образом, на вход каждого автомата поступает символ, далее из таблицы действий выбирается и выполняется действие, которое находится в столбце текущего состояния и строке входящего символа. После этого из таблицы состояний выбирается и устанавливается как текущее то состояние, которое находится на пересечении тех же столбца и строки.

Каждое действие в таблице действий автомата осуществляет различные манипуляции: создание новой структуры, обработка входящего символа и добавление его в соответствующую структуру, инициализация и запуск другого автомата для обработки соответствующей входящей структуры. Возможна ситуация, когда необходимо только изменить стостояние автомата, при этом никаких действий выполнять не требуется. В таком случае на пересечении соответствующего столбца и строки в таблице действий установлено значение *null*.

Каждый автомат среди списка состояний имеет состояние *End*, которое является признаком успешного завершения работы автомата. После перехода автомата в это состояние его работа завершается и автомат возвращает структуру с данными, которые он получил из входного потока символов. Все автоматы имеют среди списка символов символ *Eof* – признак конца файла. Для разных атоматов и разных состояний при поступлении этого символа могут выполняться разные действия: генерирование синтаксической ошибки или же переход в состояние *End*.

Большинство автоматов среди списка состояний имеют состояние *Error*, переход в которое свидетельствует о синтаксической ошибке во входном потоке. Кроме того, для каждого перехода в состояние *Error* могут выполняться различные действия из таблицы действий. Суть их одинакова – генерирование исключительной ситуации, которая сообщает об ошибке, однако для каждого типа ошибки указываются различные текстовые сообщения об ошибке, то есть описание того, какой именно тип ошибки был допущен в текущей позиции входного потока символов. Таким образом, для каждого автомата имеется массив методов *Error*, а в ячейках таблицы действий укзывется элемент этого массива *Error[i].* Данный подход позволяет не только идентифицировать, но и классифицировать синтаксические ошибки.

Рассмотрим более подробно процесс анализа каждого типа синтаксиса.

**Разбор синтаксиса адресов электронной почты**

Инфраструктура состоит из единственного автомата.

Список символов входного потока:

public enum Symbol

{

Separator, // Space, Tab, New line

At, // @

Dot, // .

LetrDigit, // Letter, Digit

Underline, // \_

Minus, // -

Eof, // End of file

Other // Other symbols

}

Список состояний автомата:

public enum State

{

Ready, // No emails processing now

Name, // Name processing

DomBegin, // Begin of domain or subdomain

Domain, // Domain processing

Minus, // Current symbol is minus

End, // Parsing done

Error // Error occured

}

Таблица переходов атомата:

static readonly State[,] TransitionTable =

{ // Ready Name DomBegin Domain Minus End Error

{State.Ready, State.Error, State.Error, State.Ready, State.Error, State.End, State.Error}, // Separator

{State.Error, State.DomBegin, State.Error, State.Error, State.Error, State.End, State.Error}, // At

{State.Error, State.Error, State.Error, State.DomBegin, State.Error, State.End, State.Error}, // Dot

{State.Name, State.Name, State.Domain, State.Domain, State.Domain, State.End, State.Error}, // LetrDigit

{State.Name, State.Name, State.Error, State.Error, State.Error, State.End, State.Error}, // Underline

{State.Error, State.Error, State.Error, State.Domain, State.Domain, State.End, State.Error}, // Minus

{State.End, State.Error, State.Error, State.End, State.Error, State.End, State.Error}, // Eof

{State.Error, State.Error, State.Error, State.Error, State.Error, State.End, State.Error} // Other

};

Таблица действий автомата:

static readonly Method[,] ActionTable =

{ // Ready Name DomBegin Domain Minus End Error

{null, Error[5], Error[7], null, Error[8], null, Error[0]}, // Separator

{Error[1], AddDomain, Error[4], Error[4], Error[4], null, Error[0]}, // At

{Error[2], Error[2], Error[4], AddDomain, Error[9], null, Error[0]}, // Dot

{NewEmail, AppendName, AppendDomain, AppendDomain, AppendDomain, null, Error[0]}, // LetrDigit

{NewEmail, AppendName, Error[8], Error[8], Error[8], null, Error[0]}, // Underline

{Error[3], Error[3], Error[9], AppendDomain, AppendDomain, null, Error[0]}, // Minus

{null, Error[6], Error[6], null, Error[9], null, Error[0]}, // Eof

{Error[4], Error[4], Error[4], Error[4], Error[4], null, Error[0]} // Other

};

Результатом работы автомата является список объектов следующего класса:

/// <summary>

/// Email structure

/// </summary>

public class Email

{

public string Name = string.Empty;

public List<string> Domain = new List<string>(); // List of subdomains

}

**Разбор синтаксиса секций, параметров, комментариев, строк**

Инфраструктура состоит из главного автомата, задачей которого является начальная идентификация типа структуры во входящем потоке, после чего управление передаётся другому автомату, который отвечает за разбор уже конкретной свойственной ему структуры из входного потока. Главный автомат собирает все структуры, которые возвращают другие автоматы, в общий список объектов.

Были разработаны следующие типы автоматов:

* главный автомат;
* автомат для разбора комментариев, начинающихся символом «#»;
* автомат для разбора комментариев, начинающихся символами «/\*»;
* автомат для разбора строк, заключённых в двойные кавычки;
* автомат для разбора имени секции или параметра;
* автомат для разбра секции;
* автомат для разбора параметра.

Дерево создания и вызовов автоматов имеют следующий вид:

Main

#Comment

/\*Comment\*/

Name of section/parameter

Section

Parameter

“String”

“String”

Кольцевые связи в дереве указывают на то, что автомат для разбора секций может создавать и вызывать автоматы для разбора остальных структур, так секция может в себе содержать все остальные структуры, в том числе другие секции. Таким образом, структура для хранения секции является рекурсивной.

Рассмотрим свойства каждого из автоматов.

**1) Главный автомат – возвращает список элементов класс *Object*:**

Список символов входного потока:

public enum Symbol

{

Separator, // Space, Tab, New line

Slash, // /

Asterisk, // \*

Sharp, // #

Letter, // Letter

Eof, // End of file

Other // Other symbol

}

Список состояний автомата:

public enum State

{

Ready, // No structures processing now

Slash, // Current symbol is "/"

End, // Parsing done

Error // Error occured

}

Таблица переходов атомата:

static readonly State[,] TransitionTable =

{ // Ready Slash End Error

{State.Ready, State.Error, State.End, State.Error}, // Separator

{State.Slash, State.Error, State.End, State.Error}, // Slash

{State.Error, State.Ready, State.End, State.Error}, // Asterisk

{State.Ready, State.Error, State.End, State.Error}, // Sharp

{State.Ready, State.Error, State.End, State.Error}, // Letter

{State.End, State.Error, State.End, State.Ready}, // Eof

{State.Error, State.Error, State.End, State.Error} // Other

};

Таблица действий автомата:

static readonly Method[,] ActionTable =

{ // Ready Slash End Error

{null, Error, null, Error}, // Separator

{null, Error, null, Error}, // Slash

{Error, SlashComment, null, Error}, // Asterisk

{SharpComment, Error, null, Error}, // Sharp

{Name, Error, null, Error}, // Letter

{null, null, null, Error}, // Eof

{Error, Error, null, Error} // Other

};

2) Автомат для разбора комментариев, начинающихся символом «#»:

Список символов входного потока:

public enum Symbol

{

SpaceTab, // Space, Tab

NewLine, // New line

Eof, // End of file

Other // Other symbol

}

Список состояний автомата:

public enum State

{

Begin, // Waiting for comment chars

Comment, // Comment chars

End // Parsing done

}

Таблица переходов атомата:

static readonly State[,] TransitionTable =

{ // Begin Comment End

{State.Begin, State.Comment, State.End}, // SpaceTab

{State.End, State.End, State.End}, // NewLine

{State.End, State.End, State.End}, // Eof

{State.Comment, State.Comment, State.End} // Other

};

Таблица действий автомата:

static readonly Method[,] ActionTable =

{ // Begin Comment End

{null, AddChar, null}, // Separator

{null, null, null}, // NewLine

{null, null, null}, // Eof

{AddChar, AddChar, null} // Other

};

Результатом работы автомата является объект класса *String*.

3) Автомат для разбора комментариев, начинающихся символами «/\*»:

Список символов входного потока:

public enum Symbol

{

Asterisk, // \*

Slash, // /

Eof, // End of file

Other // Other symbol

}

Список состояний автомата:

public enum State

{

Comment, // Comment chars

Asterisk, // Current symbol is "\*"

End, // Parsing done

Error // Error occured

}

Таблица переходов атомата:

static readonly State[,] TransitionTable =

{ // Comment Asterisk End Error

{State.Asterisk, State.Asterisk, State.End, State.Error}, // Asterisk

{State.Comment, State.End, State.End, State.Error}, // Slash

{State.Error, State.Error, State.End, State.Error}, // Eof

{State.Comment, State.Comment, State.End, State.Error} // Other

};

Таблица действий автомата:

static readonly Method[,] ActionTable =

{ // Comment Asterisk End Error

{null, AddChar, null, Error}, // Asterisk

{AddChar, null, null, Error}, // Slash

{Error, Error, null, Error}, // Eof

{AddChar, AddAsterAndChr, null, Error} // Other

};

Результатом работы автомата является объект класса *String*.

4) Автомат для разбора строк, заключённых в двойные кавычки:

Список символов входного потока:

public enum Symbol

{

Quoting, // "

Slash, // \

n, // n

t,

SingleQuoting, // '

Eof, // End of file

Other // Other symbol

}

Список состояний автомата:

public enum State

{

Common, // String regular chars

Slash, // Current char is "/"

End, // Parsing done

Error // Error occured

}

Таблица переходов атомата:

public static readonly State[,] TransitionTable =

{ // Common Slash End Error

{State.End, State.Common, State.End, State.Error}, // Quoting

{State.Slash, State.Common, State.End, State.Error}, // Slash

{State.Common, State.Common, State.End, State.Error}, // n

{State.Common, State.Common, State.End, State.Error}, // t

{State.Common, State.Common, State.End, State.Error}, // SingleQuoting

{State.Error, State.Error, State.End, State.Error}, // Eof

{State.Common, State.Error, State.End, State.Error} // Other

};

Таблица действий автомата:

public static readonly Method[,] ActionTable =

{ // Common Slash End Error

{null, AddQuoting, null, Error[0]}, // Quoting

{null, AddSlash, null, Error[0]}, // Slash

{AddChar, AddNewLine, null, Error[0]}, // n

{AddChar, AddTab, null, Error[0]}, // t

{AddChar, AddSingleQuote, null, Error[0]}, // SingleQuoting

{Error[1], Error[1], null, Error[0]}, // Eof

{AddChar, Error[2], null, Error[0]} // Other

};

Результатом работы автомата является объект класса *String*, в котором особые символы заменяются на их названия.

5) Автомат для разбора имени секции или параметра:

Список символов входного потока:

public enum Symbol

{

Separator, // Space, Tab, New line

LtrDgtUnd, // Letter, Digit, \_

Equality, // =

Eof, // End of file

Other // Other symbol

}

Список состояний автомата:

public enum State

{

Name, // Current symbol is name part

Separator, // Current symbol is Space, Tab or New line

End, // Parsing done

Error, // Error occured

}

Таблица переходов атомата:

static readonly State[,] TransitionTable =

{ // Name Separator End Error

{State.Separator, State.Separator, State.End, State.Error}, // Separator

{State.Name, State.End, State.End, State.Error}, // LtrDgtUnd

{State.End, State.End, State.End, State.Error}, // Equality

{State.Error, State.Error, State.End, State.Error}, // Eof

{State.Error, State.End, State.End, State.Error}, // Other

};

Таблица действий автомата:

static readonly Method[,] ActionTable =

{ // Name Separator End Error

{null, null, null, Error[0]}, // Separator

{AddChar, ReturnSection, null, Error[0]}, // LtrDgtUnd

{ReturnParameter, ReturnParameter, null, Error[0]}, // Equality

{Error[1], Error[1], null, Error[0]}, // Eof

{Error[2], ReturnSection, null, Error[0]}, // Other

};

Результатом работы автомата является объект класса *Parameter* или объект класса *Section*.

**6) Автомат для разбора параметра:**

Список символов входного потока:

public enum Symbol

{

Separator, // Space, Tab, New line

Quoting, // "

Semicolon, // ;

Eof, // End of file

Other // Other symbol

}

Список состояний автомата:

public enum State

{

Begin, // Begin of parameter value

Value, // Parameter value

End, // Parsing done

Error // Error occured

}

Таблица переходов атомата:

public static readonly State[,] TransitionTable =

{ // Begin Parameter End Error

{State.Begin, State.Value, State.End, State.Error}, // Separator

{State.Value, State.Value, State.End, State.Error}, // Quoting

{State.End, State.End, State.End, State.Error}, // Semicolon

{State.End, State.Error, State.End, State.Error}, // Eof

{State.Value, State.Value, State.End, State.Error} // Other

};

Таблица действий автомата:

public static readonly Method[,] ActionTable =

{ // Begin Parameter End Error

{null, AddChar, null, Error}, // Separator

{AddString, AddString, null, Error}, // Quoting

{null, null, null, Error}, // Semicolon

{Error, Error, null, Error}, // Eof

{AddChar, AddChar, null, Error} // Other

};

Результатом работы автомата является объект следующего класса:

/// <summary>

/// Parameter structure

/// </summary>

class Parameter

{

public string Name = string.Empty; // Parameter name

public string Value = string.Empty; // Parameter value

}

**7) Автомат для разбра секции:**

Список символов входного потока:

public enum Symbol

{

LeftBracket, // {

Separator, // Space, Tab, New line

Slash, // /

Asterisk, // \*

Sharp, // #

Letter, // Letter

RightBracket, // }

Eof, // End of file

Other // Other symbol

}

Список состояний автомата:

public enum State

{

Value, // Section value

Inside, // Inside section

Slash, // Current symbol is "/"

End, // Parsing done

Error // Error occured

}

Таблица переходов атомата:

static readonly State[,] TransitionTable =

{ // Value Inside Slash End Error

{State.Inside, State.Error, State.Error, State.End, State.Error}, // LeftBracket

{State.Value, State.Inside, State.Error, State.End, State.Error}, // Separator

{State.Value, State.Slash, State.Error, State.End, State.Error}, // Slash

{State.Value, State.Error, State.Inside, State.End, State.Error}, // Asterisk

{State.Value, State.Inside, State.Error, State.End, State.Error}, // Sharp

{State.Value, State.Inside, State.Error, State.End, State.Error}, // Letter

{State.Error, State.End, State.Error, State.End, State.Error}, // Right Bracket

{State.Error, State.Error, State.Error, State.End, State.Error}, // Eof

{State.Value, State.Error, State.Error, State.End, State.Error} // Other

};

Таблица действий автомата:

static readonly Method[,] ActionTable =

{ // Value Inside Slash End Error

{null, Error[3], Error[5], null, Error[0]}, // LeftBracket

{AddChar, null, Error[5], null, Error[0]}, // Separator

{AddChar, null, Error[5], null, Error[0]}, // Slash

{AddChar, Error[4], SlashComment, null, Error[0]}, // Asterisk

{AddChar, SharpComment, Error[5], null, Error[0]}, // Sharp

{AddChar, Name, Error[5], null, Error[0]}, // Letter

{Error[2], null, Error[5], null, Error[0]}, // Right Bracket

{Error[1], Error[1], Error[1], null, Error[0]}, // Eof

{AddChar, Error[4], Error[5], null, Error[0]} // Other

};

Результатом работы автомата является объект следующего класса:

/// <summary>

/// Section structure - the same as parameter but also stores another objects inside

/// </summary>

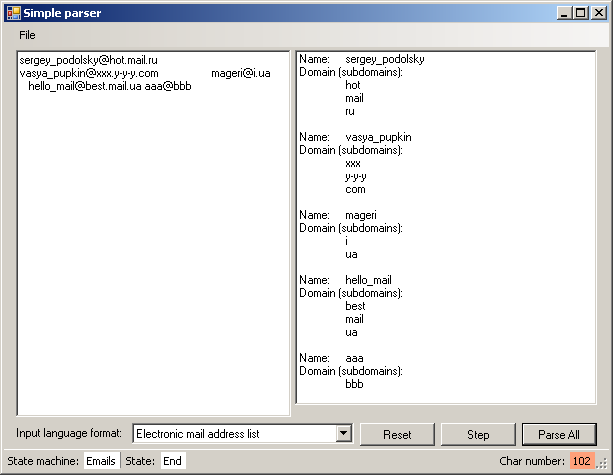
class Section : Parameter

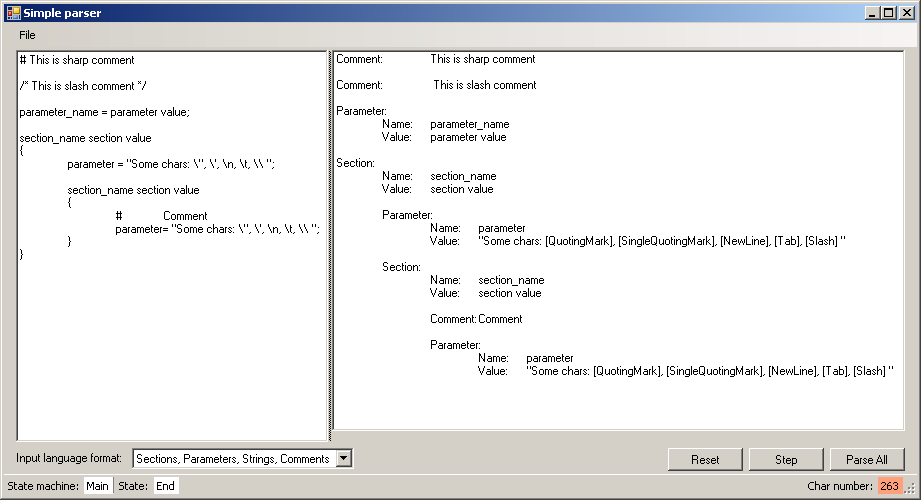
{

public List<object> Items = new List<object>(); // List of objects that section contains inside

}

1. **Примеры работы парсера:**

****

****