1 Цель работы

Целью данной работы является приобретение навыка разработки простейших лексических анализаторов, работающих на основе поиска в тексте по образцу, заданному регулярным выражением.

2 Исходные данные

Стандартная библиотека любого современного языка программирования содержит средства для поиска в тексте образцов, заданных регулярными выражениями. При этом используется расширенный синтаксис записи регулярных выражений, позволяющий по сути выйти за рамки регулярных языков. Механизм поиска по таким регулярным выражениям годится для написания простейших лексических анализаторов. Однако, для этого механизма характерна нелинейная зависимость времени работы от длины распознаваемой лексемы, поэтому в промышленных компиляторах он не используется.

В качестве языка реализации в данной лабораторной работе выберем язык Java, стандартная библиотека которого содержит пакет `java.util.regex`, в котором располагаются классы `Pattern` и `Matcher`, предназначенные для поиска по регулярным выражениям. Документация по этому пакету находится по адресу:

http://docs.oracle.com/javase/7/docs/api/java/util/regex/package-summary.html.

Вводную статью по синтаксису регулярных выражений можно прочитать здесь:

http://www.quizful.net/post/Java-RegExp.

Идея лексического анализа на основе поиска по регулярным выражениям состоит в использовании групп, представляющих собой фрагменты регулярных выражений, заключённые в круглые скобки, значения которых запоминаются при сопоставлении текста с образцом. Например, на листинге ниже показано, как с использованием групп отличить идентификаторы от числовых литералов.

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

// Компиляция для Windows:
// javac -encoding utf8 IdentVsNumber.java

public class IdentVsNumber
{
    public static void main(String args[])
    {
        test_match("Альфа123");
        test_match("42");
    }

    public static void test_match(String text)
    {
        // Регулярные выражения
        String ident = "\\p{L}[\\p{L}0-9]*";
        String number = "[0-9]+";
        String pattern = "(?<ident>^"+ident+")|(?<number>^"+number+")";
        // Компиляция регулярного выражения
        Pattern p = Pattern.compile(pattern);
        // Сопоставление текста с регулярным выражением
        Matcher m = p.matcher(text);
        if (m.find()) {
            if (m.group("ident") != null) {
                System.out.println("Идентификатор " + m.group("ident"));
            } else {
                System.out.println("Число " + m.group("number"));
            }
        } else {
            System.out.println("Ошибка");
        }
    }
}
```

3 Задание

В лабораторной работе необходимо реализовать на языке Java две первые фазы стадии анализа: чтение входного потока и лексический анализ. Чтение входного потока должно осуществляться из файла (в UTF-8), при этом лексический анализатор должен вычислять текущие координаты в обрабатываемом тексте. В результате работы программы в стандартный поток вывода должны выдаваться описания распознанных лексем в формате

```
Тег (координаты): значение
```

Например,

```
IDENT (1, 2): count
ASSIGN (1, 8): :=
NUMBER (1, 11): 100
```

Лексемы во входном файле могут разделяться пробельными символами (пробел, горизонтальная табуляция, маркеры окончания строки), а могут быть записаны слитно (если это не приводит к противоречиям).

Идентификаторы и числовые литералы не могут содержать внутри себя пробельных символов, если в задании явно не указано иного. Комментарии, строковые и символьные литералы могут содержать внутри себя пробельные символы.

Входной файл может содержать ошибки, при обнаружении которых лексический анализатор должен выдавать сообщение с указанием координаты:

```
syntax error (10,2)
```

После обнаружения ошибки лексический анализатор должен восстанавливаться по следующей схеме: из входного потока пропускаются все подряд идущие символы до нахождения следующей лексемы.

Лексический анализатор должен иметь программный интерфейс для взаимодействия с парсером. Рекомендуется реализовывать его как итератор с методом `nextToken()` для императивных языков или функцию, возвращающую список токенов, для функциональных языков.

В регулярных выражениях рекомендуется использовать классы символов Unicode для обозначения букв, чисел и других подобных множеств. Многие движки регулярных выражений для задания классов используют синтаксис `\p{класс}`. Вместо нумерованных групп рекомендуется использовать именованные (`?<имя>regexp`), при использовании нумерованных групп ненумеруемые обозначаются как (`?:regexp`).

4 Индивидуальный вариант

Строковые литералы: органичены двойными кавычками, для включения двойной кавычки она удваивается, для продолжения литерала на следующей строчке текста в конце текущей строчки ставится знак «\». Целочисленные литералы: либо последовательности десятичных цифр, либо последовательности шестнадцатеричных цифр (в любом регистре), начинающиеся с «$». Идентификаторы: последовательности буквенных символов Unicode, цифр и знаков «$», начинающиеся с буквы.

5 Отчёт по лабораторной работе № 1.2

Отчёт выполняется в разметке Markdown по следующему шаблону:

```
% Лабораторная работа № 1.2. «Лексический анализатор
  на основе регулярных выражений»
% 14 марта 2023 г.
% Вася Пупкин, ИУ9-63Б
# Цель работы
‹переписываете цель работы из задания›
# Индивидуальный вариант
‹переписываете индивидуальный вариант›
# Реализация
```java
…
```
Тестирование
Входные данные
```

…
```
Вывод на `stdout` (если необходимо)
```

…
```
# Вывод
‹пишете, чему научились›
```

В отчёте приведён лишь необходимый минимум.

Шаблон отчёта

Ваш отчёт будет конвертирован в PDF при помощи pandoc следующей командой:

```
pandoc \
  --pdf-engine=xelatex \
  -V 'mainfont:Liberation Serif' \
  -V 'monofont:Liberation Mono' \
  "$SOURCE" -o "$PDF"
```

Язык реализации: Markdown

Код решения

Из файла

Отправить