

# Лабораторная работа №2. Ручное построение нисходящих синтаксических анализаторов

26 апреля 2011 г.

## Введение

Цель данной лабораторной работы — научиться разрабатывать грамматики для заданных неформально языков с учетом семантического смысла и приоритета операторов, разрабатывать лексические анализаторы, разрабатывать ручную нисходящие синтаксические анализаторы, разрабатывать подробные тестовые наборы для анализаторов.

Форма отчетности: программа и текстовый отчет. Программа может быть написана на любом языке программирования общего назначения (рекомендуются Си++ и Java).

## 1 Разработка грамматики

Разработайте контекстно-свободную грамматику для языка, описанного в условии вашего варианта. Сначала разработайте грамматику, исходя из структуры языка, чтобы она максимально близко соответствовала интуитивным представлениям о построении слов из языка. Затем, при необходимости, устраните левую рекурсию и/или правое ветвление.

В отчете приведите исходную и преобразованную грамматику, опишите смысл всех нетерминалов.

## Пример задания

*Язык правильных скобочных выражений*

### Пример выполнения задания

Построим грамматику.

$$S \rightarrow SS$$

$$S \rightarrow (S)$$

$$S \rightarrow \varepsilon$$

Нетерминал	Описание
$S$	Правильная скобочная последовательность.

В грамматике есть левая рекурсия. Устраним ее. Получится грамматика:

$$S \rightarrow (S)S'$$

$$S \rightarrow \varepsilon$$

$$S' \rightarrow SS'$$

$$S' \rightarrow \varepsilon$$

Нетерминал	Описание
$S$	Правильная скобочная последовательность.
$S'$	Продолжение правильной скобочной последовательности.

## 2 Построение лексического анализатора

Лексический анализатор должен получать на вход строку и выдавать последовательность терминалов (токенов). Пробелы и переводы строк должны игнорироваться.

### Пример выполнения задания

В нашей грамматике два терминала: '(' и ')'.  
Построим лексический анализатор. Заведем класс `Token` для хранения терминалов. Не забудем также про конец строки.

```
public enum Token {
    LPAREN, RPAREN, END
}
```

Терминал	Токен
(	LPAREN
)	RPAREN
\$	END

Ниже приведен пример простейшего лексического анализатора. Если токены в вашей грамматике могут состоять из нескольких символов, это следует учесть.

```
import java.io.IOException;
import java.io.InputStream;
import java.text.ParseException;

public class LexicalAnalyzer {
    InputStream is;
    int curChar;
    int curPos;
    Token curToken;

    public LexicalAnalyzer(InputStream is) throws ParseException {
        this.is = is;
        curPos = 0;
        nextChar();
    }

    private boolean isBlank(int c) {
        return c == ' ' || c == '\r' || c == '\n' || c == '\t';
    }

    private void nextChar() throws ParseException {
        curPos++;
        try {
            curChar = is.read();
        } catch (IOException e) {
            throw new ParseException(e.getMessage(), curPos);
        }
    }
}
```

```

public void nextToken() throws ParseException {
    while (isBlank(curChar)) {
        nextChar();
    }
    switch (curChar) {
        case '(':
            nextChar();
            curToken = Token.LPAREN;
            break;
        case ')':
            nextChar();
            curToken = Token.RPAREN;
            break;
        case -1:
            curToken = Token.END;
            break;
        default:
            throw new ParseException("Illegal_character_"
                + (char) curChar, curPos);
    }
}

public Token curToken() {
    return curToken;
}

public int curPos() {
    return curPos();
}
}

```

### 3 Построение синтаксического анализатора

Постройте множества FIRST и FOLLOW для нетерминалов вашей грамматики. Затем напишите синтаксический анализатор с использованием рекурсивного спуска.

## Пример выполнения задания

Построим множества FIRST и FOLLOW для нетерминалов нашей грамматики.

Нетерминал	FIRST	FOLLOW
$S$	(, $\varepsilon$	), \$
$S'$	(, $\varepsilon$	), \$

Заведем структуру данных для хранения дерева.

```
import java.util.Arrays;
import java.util.List;

public class Tree {
    String node;

    List<Tree> children;

    public Tree(String node, Tree... children) {
        this.node = node;
        this.children = Arrays.asList(children);
    }

    public Tree(String node) {
        this.node = node;
    }
}
```

Ниже приведен простейший синтаксический анализатор с использованием рекурсивного спуска.

```
import java.io.InputStream;
import java.text.ParseException;

public class Parser {
    LexicalAnalyzer lex;

    Tree S() throws ParseException {
        switch (lex.curToken()) {
            case LPAREN:
                // (
                lex.nextToken();
```

```

        // S
        Tree sub = S();
        // )
        if (lex.curToken() != Token.RPAREN) {
            throw new ParseException("_expected_" +
                "at_position", lex.curPos());
        }
        lex.nextToken();
        // S'
        Tree cont = SPrime();
        return new Tree("S", new Tree("("), sub,
            new Tree(")"), cont);
    case RPAREN:
    case END:
        // eps
        return new Tree("S");
    default:
        throw new AssertionError();
    }
}

```

```

Tree SPrime() throws ParseException {
    switch (lex.curToken()) {
        case LPAREN:
            // S
            Tree sub = S();
            // S'
            Tree cont = SPrime();
            return new Tree("S'", sub, cont);
        case RPAREN:
        case END:
            // eps
            return new Tree("S'");
        default:
            throw new AssertionError();
    }
}

```

```

Tree parse(InputStream is) throws ParseException {
    lex = new LexicalAnalyzer(is);
}

```

```

        lex.nextToken();
        return S();
    }
}

```

## 4 Визуализация дерева разбора

Для изучения результата разработайте систему визуализации дерева разбора.

## 5 Подготовка набора тестов

Подготовьте набор исчерпывающих тестов для вашей программы. По возможности добейтесь, чтобы все возможные ошибки в грамматике или в лексическом/синтаксическом анализаторе ими покрывались. Про каждый тест дайте короткое описание.

### Пример выполнения задания

Ниже приведен лишь небольшой набор тестов, исчерпывающий набор для правильных скобочных последовательностей должен содержать больше тестов.

Тест	Описание
()()	Тест на правило $S \rightarrow SS$
(( ))	Тест на правило $S \rightarrow (S)$
	Тест на правило $S \rightarrow \varepsilon$
((()((()))()()))()	Небольшой случайный тест

## Задания

### Вариант 1. Арифметические выражения

Арифметические выражения с операциями сложения, вычитания, умножения, скобками, унарным минусом. Приоритет операций стандартный.

В качестве операндов выступают целые числа. Используйте один терминал для всех чисел.

Пример:  $(1+2)*(-3*(7-4)+2)$

## Вариант 2. Регулярные выражения

Регулярные выражения с операциями конкатенации (простая последовательная запись строк), выбора (вертикальная черта), замыкания Клини. Приоритет операций стандартный. Скобки могут использоваться для изменения приоритета.

Для обозначения базовых языков используются маленькие буквы латинского алфавита. Используйте один терминал для всех символов.

Пример:  $((abc*b|a)*ab(aa|b*)b)*$

## Вариант 3. Арифметические выражения в постфиксной записи

Арифметические выражения в постфиксной записи. Используются операции сложения, вычитания, умножения.

В качестве операндов выступают целые числа. Используйте один терминал для всех чисел.

Пример: 1 2 10 + \*

## Вариант 4. Арифметические выражения в префиксной записи

Арифметические выражения в постфиксной записи. Используются операции сложения, вычитания, умножения.

В качестве операндов выступают целые числа. Используйте один терминал для всех чисел.

Пример: \* + 2 2 + 3 1

## Вариант 5. Логические формулы в стиле Паскаля

Логические формулы. Используются операции `and`, `or`, `xor`, `not`. Приоритет операций стандартный. Скобки могут использоваться для изменения приоритета.

В качестве операндов выступают переменные с именем из одной буквы. Используйте один терминал для всех переменных. Для каждой логической операции должен быть заведен один терминал (не три 'a', 'n', 'd' для `and`).

Пример: (a and b) or not (c xor (a or not b))



## Вариант 6. Логические формулы в стиле Си

Логические формулы. Используются операции `&`, `|`, `^`, `!`. Приоритет операций стандартный. Скобки могут использоваться для изменения приоритета.

В качестве операндов выступают переменные с именем из одной буквы. Используйте один терминал для всех переменных. Для каждой логической операции должен быть заведен один терминал.

Пример: `(!a | b) & a & (a | !(b ^ c))`

## Вариант 7. Описание переменных в Паскале

Блок описания переменных в Паскале. Блок начинается ключевым словом `var`, далее идет несколько описаний переменных. Описание переменных содержит имена переменных через запятую, затем двоеточие, затем имя типа. Достаточно рассматривать только примитивные типы (массивы, записи и т. п. не требуется).

Используйте один терминал для всех имен переменных и имен типов. Используйте один терминал для ключевого слова `var` (не три `'v'`, `'a'`, `'r'`).

Пример: `var a, b: integer; c: integer;`

## Вариант 8. Описание переменных в Си

Описания переменных в Си. Сначала следует имя типа, затем разделенные запятой имена переменных. Переменная может быть указателем, в этом случае перед ней идет звездочка (возможны и указатели на указатели, и т. д.). Описаний может быть несколько.

Используйте один терминал для всех имен переменных и имен типов.

Пример: `int a, *b, **c, d;`