Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ имени Н.Э.БАУМАНА
(национальный исследовательский университет)»

Факультет: Информатика и системы управления

Кафедра: Теоретическая информатика и компьютерные технологии

Лабораторная работа № 5

Раскрутка самоприменимого компилятора

Вариант 2

Работу выполнил

студент группы ИУ9-61

Бакланова А.Д.

Москва, 2020

Цель работы:

Целью данной работы является изучение использования детерминированных конечных автоматов с размеченными заключительными состояниями (лексических распознавателей) для решения задачи лексического анализа.

Исходные данные:

Пусть лексическая структура модельного языка состоит из шести лексических доменов:

1. пробелы — непустые последовательности пробельных символов (пробел, горизонтальная табуляция, маркеры конца строки);
2. идентификаторы — непустые последовательности латинских букв и десятичных цифр, начинающиеся с буквы;
3. целочисленные литералы — непустые последовательности десятичных цифр;
4. ключевые слова (варианты ключевых слов перечислены в таблицах 1, 2 и 3);
5. знаки операций (варианты знаков операций перечислены в таблицах 1, 2 и 3);
6. комментарии или строковые литералы (варианты перечислены в таблицах 1, 2 и 3).

Чтобы не усложнять лексический анализатор, разрешим идентификаторам примыкать справа к целочисленным литералам.

Индивидуальный вариант:

2	`do`, `while`, `<`, `>`, `<>`, строковые литералы ограничены апострофами, не могут пересекать границы строк текста.

Задание:

Выполнение лабораторной работы состоит из пяти этапов:

1. описание лексических доменов модельного языка в виде регулярных выражений;
2. построение недетерминированного лексического распознавателя для модельного языка;
3. детерминизация построенного лексического распознавателя и факторизация его алфавита;

4. построение массива обобщённых символов, матрицы переходов и массива заключительных состояний для полученного детерминированного лексического распознавателя с факторизованным алфавитом;

5. разработка лексического анализатора, работающего на основе интерпретации построенных структур данных.

Входной поток для лексического анализатора должен загружаться из файла (в ASCII). В результате работы программы в стандартный поток вывода должны выдаваться описания распознанных лексем в формате

```
Тег (координаты_фрагмента): изображение_лексемы
```

При этом лексемы, принадлежащие домену пробелов, должны отбрасываться.

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

Входной файл может содержать ошибки, при обнаружении которых лексический анализатор должен выдавать сообщение с указанием координаты, восстанавливаться и продолжать работу.

В лабораторной работе разрешается использовать любой язык программирования, поддерживающий массивы с операцией доступа к элементу по индексу, работающей за константное время.

Реализация:

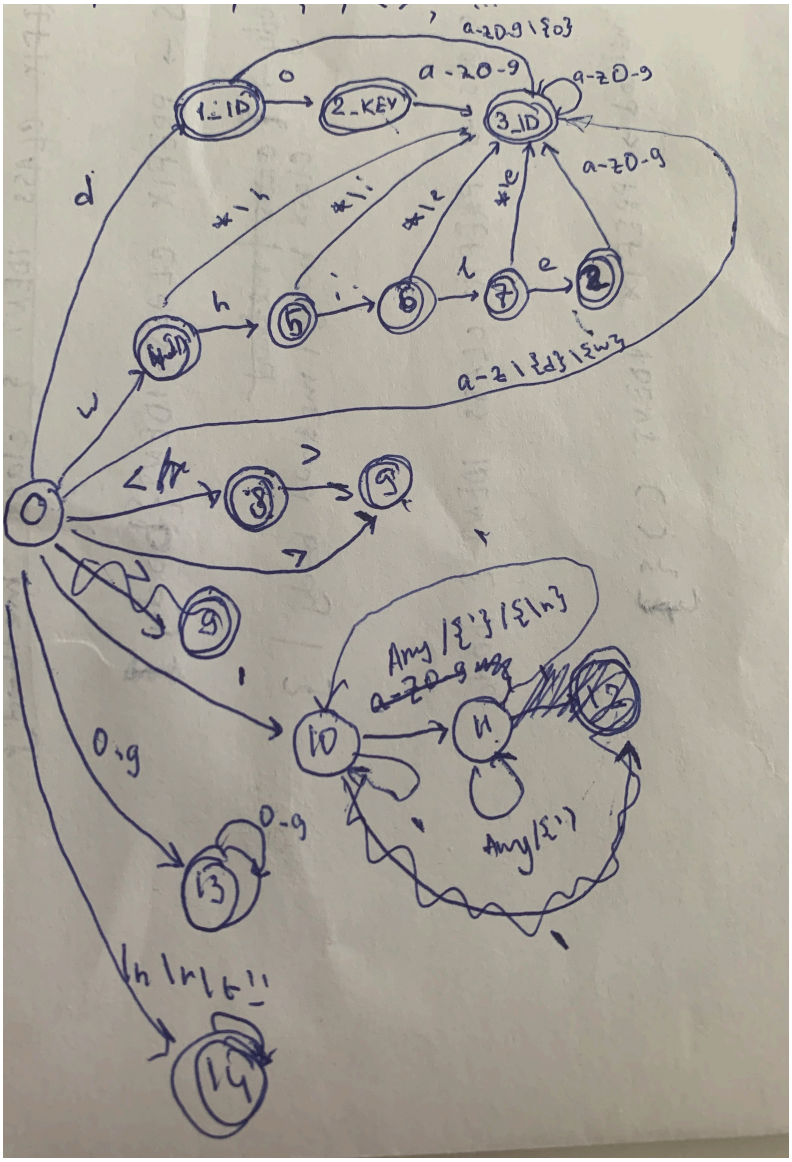
```
    public final static int[][] table = {
                          /*  d   o   w   h   i   l   e    <    >    '   num  ws a-zA-z EOL  */
                /*  START   */{ 1,  3,  4,  3,  3,  3,  3,   8,   9,  10,  13,  14,  3,  14,  -1},
                /*  ID_1    */{ 3,  2,  3,  3,  3,  3,  3,  -1,  -1,  -1,   3,  -1,  3,  -1,  -1},
                /*  KEY_2   */{ 3,  3,  3,  3,  3,  3,  3,  -1,  -1,  -1,   3,  -1,  3,  -1,  -1},
                /*  ID_3    */{ 3,  3,  3,  3,  3,  3,  3,  -1,  -1,  -1,   3,  -1,  3,  -1,  -1},
                /*  ID_4    */{ 3,  3,  3,  5,  3,  3,  3,  -1,  -1,  -1,   3,  -1,  3,  -1,  -1},
                /*  ID_5    */{ 3,  3,  3,  3,  6,  3,  3,  -1,  -1,  -1,   3,  -1,  3,  -1,  -1},
                /*  ID_6    */{ 3,  3,  3,  3,  3,  7,  3,  -1,  -1,  -1,   3,  -1,  3,  -1,  -1},
                /*  ID_7    */{ 3,  3,  3,  3,  3,  3,  2,  -1,  -1,  -1,   3,  -1,  3,  -1,  -1},
                /*  OP_8    */{-1, -1, -1, -1, -1, -1, -1,  -1,   9,  -1,  -1,  -1, -1,  -1,  -1},
                /*  OP_9    */{-1, -1, -1, -1, -1, -1, -1,  -1,  -1,  -1,  -1,  -1, -1,  -1,  -1},
                /*  LIT_10  */{11, 11, 11, 11, 11, 11, 11,  11,  11,  12,  11,  11, 11,  -1,  11},
                /*  LIT_11  */{11, 11, 11, 11, 11, 11, 11,  11,  11,  12,  11,  11, 11,  -1,  11},
                /*  LIT_12  */{-1, -1, -1, -1, -1, -1, -1,  -1,  -1,  -1,  -1,  -1, -1,  -1,  -1},
                /*  NUM_13  */{-1, -1, -1, -1, -1, -1, -1,  -1,  -1,  -1,  13,  -1, -1,  -1,  -1},
                /*  WS_14   */{-1, -1, -1, -1, -1, -1, -1,  -1,  -1,  -1,  -1,  14, -1,  -1,  -1},

    };
```

```
    private int getCode(char c) {
        if (c >= '0' && c <= '9')
            return 10;
        if ('<' == c )
            return 7;
        if ('>' == c )
            return 8;
        if (' ' == c  || '\r' == c || '\t' == c)
            return 11;
        if ('\'' == c)
            return 9;

        switch (c) {
            case 'd':
                return 0;
            case 'o':
                return 1;
            case 'w':
                return 2;
            case 'h':
                return 3;
            case 'i':
                return 4;
            case 'l':
                return 5;
            case 'e':
                return 6;
            case '\n':
                return 13;
        }

        if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
            return 12;

        return 14;
    }

@   private String getStateName(int state) {
        if (state == 1 || state == 3 || (state <= 7 && state>=4))
            return "IDENT";

        if (state == 12)
            return "STRING LITERAL";

        switch (state) {
            case 13:
                return "NUMBER";
            case 8:
                return "OPERATION";
            case 9:
                return "OPERATION";
            case 14:
                return "WHITESPACE";
            case 2:
                return "KEYWORD";
            default:
                return "ERROR";
        }
    }
```

Тестирование:

```
do
while
dow
<
>
<>
do while a77
'qqqq do while'
''a
5
'qa
 aa'
'
'
```

```
KEYWORD (1, 1)-(1, 3): do
WHITESPACE (1, 3)-(2, 1):
KEYWORD (2, 1)-(2, 6): while
WHITESPACE (2, 6)-(3, 1):
IDENT (3, 1)-(3, 4): dow
WHITESPACE (3, 4)-(4, 1):
OPERATION (4, 1)-(4, 2): <
WHITESPACE (4, 2)-(5, 1):
OPERATION (5, 1)-(5, 2): >
WHITESPACE (5, 2)-(6, 1):
OPERATION (6, 1)-(6, 3): <>
WHITESPACE (6, 3)-(7, 1):
KEYWORD (7, 1)-(7, 3): do
WHITESPACE (7, 3)-(7, 4):
KEYWORD (7, 4)-(7, 9): while
WHITESPACE (7, 9)-(7, 10):
IDENT (7, 10)-(7, 13): a77
WHITESPACE (7, 13)-(8, 1):
STRING LITERAL (8, 1)-(8, 16): 'qqqq do while'
WHITESPACE (8, 16)-(9, 1):
STRING LITERAL (9, 1)-(9, 3): ''
IDENT (9, 3)-(9, 4): a
WHITESPACE (9, 4)-(10, 1):
NUMBER (10, 1)-(10, 2): 5
WHITESPACE (10, 2)-(11, 1):
ERROR (11, 1)-(11, 4): 'qa
WHITESPACE (11, 4)-(12, 2):
IDENT (12, 2)-(12, 4): aa
ERROR (12, 4)-(12, 5): '
WHITESPACE (12, 5)-(13, 1):
ERROR (13, 1)-(13, 2): '
WHITESPACE (13, 2)-(14, 1):
ERROR (14, 1)-(14, 2): '
END_OF_PROGRAM (14, 2)-(14, 2):
```



Вывод:

В результате выполнения лабораторной работы были получены знания об использовании детерминированных конечных автоматов с размеченными заключительными состояниями (лексических распознавателей) для решения задачи лексического анализа.