МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт Компьютерных Наук и Кибербезопасности

Высшая школа технологий искуственного интелекта

Направление 02.03.01 Математика и Компьютерные Науки

Лабораторная работа №1

По дисциплине:
«Математическая логика и теория автоматов»

Тема работы:

«Реализация лексического анализатора»
«Вариант 14»

Обучающийся: _____ Черепанов Михаил Дмитриевич

Руководитель: _____ Востров Алексей Владимирович

«____» _____ 20__ г.

Санкт-Петербург 2024

Содержание

Введение

В данной работе требуется реализовать программу, которая выполняет лексический анализ входного текста в соответствии с заданием и порождает таблицу лексем с указанием их типов и значений. Текст на входном языке задается в виде символьного (текстового) файла. Программа должна выдавать сообщения о наличие во входном тексте ошибок, которые могут быть обнаружены на этапе лексического анализа. Длина идентификатора и строковых констант ограничена 16 символами. Программа должна допускать наличие комментариев неограниченной длины во входном файле (форма комментариев выбирается самостоятельно).

В моем варианте введены следующие правила языка:

1. Входной язык содержит операторы условия while ... do и do ... while, разделенные символом ; (точка с запятой).

2. Операторы условия содержат идентификаторы, знаки сравнения $<$, $>$, $=$, вещественные числа, знак присваивания ($:=$). Вещественные числа могут начинаться с точки.

1 Математическое описание

1.1 Конечный автомат распознаватель

Конечный автомат - абстрактная модель вычислителя, число возможных состояний которого ограничено. Формально представляется следующим образом:
$KA = (S, s_0, X, Y, \lambda, \sigma)$, где:

- S - конечное множество состояний;
- $s_0 \in S$ - начальное состояние.
- X, Y - конечные входной и выходной алфавиты;
- $\lambda : S \times X \to S$ - функция перехода в другое состояние;
- $\sigma : S \times X \to Y$ - выходные значения при переходе;

Конечный автомат начинает работу в s0 и далее может реагировать на входные воздействия, производя вычисления. В данном случае - распознавать лексемы.

1.2 Описание разработанного автомата

В данной работе:
Входной алфавит X :
Символы кодировки ASCII.
Выходной алфавит Y :
Буквенные и циферные символы, а так же символы:
{ . : = < > };
Функция выходных значений:
В качестве функции выходных значений была выбрана функция P, которая может быть функцией одной из двух следующих типов:
P(ch) - записывает текущий символ в результирующую строку.
P(LexsemType, str) - записывает текущую результирующую строку в итоговую последовательность.
Формально:
$P = P(ch) \ \ or \ \ P(LexsemType, str)$, где
$ch \in \Sigma$, Σ - входной алфавит
$P(ch) = str + ch$, str - строка, элемент дополнительной памяти, значение лексемы
$P(LexsemType, str) = st + pair(LexsemType, str)$, st - стэк для записи лексем, элемент дополнительной памяти
$LexsemType \in \{Key\_word, Compare\_operator, Assign, Number, Variable, Semicolon\}$
Функция переходов:
Функция переходов представленна в виде таблицы на рис.1- 2

		'd'	'o'	'w'	'h'	'i'	'l'	'e'	'#'	':'	leter	
1	S0	S_do1	S_var	S_w1	S_var	S_var	S_var	S_var	S_comment	S_assign	S_var	
2	S_d1	S_var	S_do_final	S_var	S_var	S_var	S_var	S_var	S_comment	S_assign	S_var	
3	S_do_final	S_var	S_var	S_var	S_var	S_var	S_var	S_var	S_comment	S_assign	S_var	
4	S_w1	S_var	S_var	S_var	S_w2	S_var	S_var	S_var	S_comment	S_assign	S_var	
5	S_w2	S_var	S_var	S_var	S_var	S_w3	S_var	S_var	S_comment	S_assign	S_var	
6	S_w3	S_var	S_var	S_var	S_var	S_var	S_w4	S_var	S_comment	S_assign	S_var	
7	S_w4	S_var	S_var	S_var	S_var	S_var	S_var	S_while_final	S_comment	S_assign	S_var	
8	S_while_final	S_var	S_var	S_var	S_var	S_var	S_var	S_var	S_comment	S_assign	S_var	
9	S_val1	S_error	S_error	S_error	S_err	S_error	S_erro	S_error	S_comment	S_assign	S_error	
10	S_val2	S_error	S_error	S_error	S_err	S_error	S_erro	S_error	S_comment	S_assign	S_error	
11	S_var	S_var	S_var	S_var	S_var	S_var	S_var	S_var	S_comment	S_assign	S_var	
12	S_comment	S_com	S_comment	S_comm	S_co	S_com	S_con	S_comment	S_comment	S_comment	S_comment	
13	S_assign	S_error	S_error	S_error	S_err	S_error	S_erro	S_error	S_comment	S_assign	S_error	
14	S_compare	S_var	S_var	S_var	S_var	S_var	S_var	S_var	S_comment	S_assign	S_var	

Рис. 1: Таблица переходов1

Digit	Space , ";"	'.'	'<'	'='	'>'	\n	other	
S_val1	S0	S_float	S_compare	S_compare	S_compare	S0	S_error	S0
S_var	S0	S_error	S_compare	S_compare	S_compare	S0	S_error	S_d1
S_var	S0	S_error	S_compare	S_compare	S_compare	S0	S_error	S_do_final
S_var	S0	S_error	S_compare	S_compare	S_compare	S0	S_error	S_w1
S_var	S0	S_error	S_compare	S_compare	S_compare	S0	S_error	S_w2
S_var	S0	S_error	S_compare	S_compare	S_compare	S0	S_error	S_w3
S_var	S0	S_error	S_compare	S_compare	S_compare	S0	S_error	S_w4
S_var	S0	S_error	S_compare	S_compare	S_compare	S0	S_error	S_while_final
S_val1	S0	S_val2	S_compare	S_compare	S_compare	S0	S_error	S_val1
S_val2	S0	S_error	S_compare	S_compare	S_compare	S0	S_error	S_val2
S_var	S0	S_error	S_compare	S_compare	S_compare	S0	S_error	S_var
S_comment	S_comment	S_comment	S_comment	S_comment	S_comment	S0	S_comment	S_comment
S_error	S0	S_error	S_error	S_0	S_error	S0	S_error	S_assign
S_val1	S0	S_val2	S_compare	S_compare	S_compare	S0	S_error	S_compare

Рис. 2: Таблица переходов2

В таблице значение leter соответсвует всем буквенным символам кроме { 'd','o','w','h','i' ,'l','e' }, представленных отдельно.

Значение digit - соответсвует цифровым символам.

Значение other - соответствует остальным символам кодировки ASCII, не представленным в таблице.

1.3 Лексема

Лексема – минимальная единица языка, имеющая смысл.

В данной работе были выделены следующие типы лексем:

Тип лексемы	Возможные значения
Key word	while / do
Compare operator	<,>,=
Assign	:=
Number	Последовательность цифр. В последовательности может присутсвовать одна точка; Последовательность может начинаться с + или -. (Длинна последовательности <=16 символов)
Variable	Последовательность букв или цифр (Начинается обязательно с буквы, длина <=16)
Semicolon	;

1.4 Граф конечного автомата

Общая схема конечного автомата представленна на рис. 3

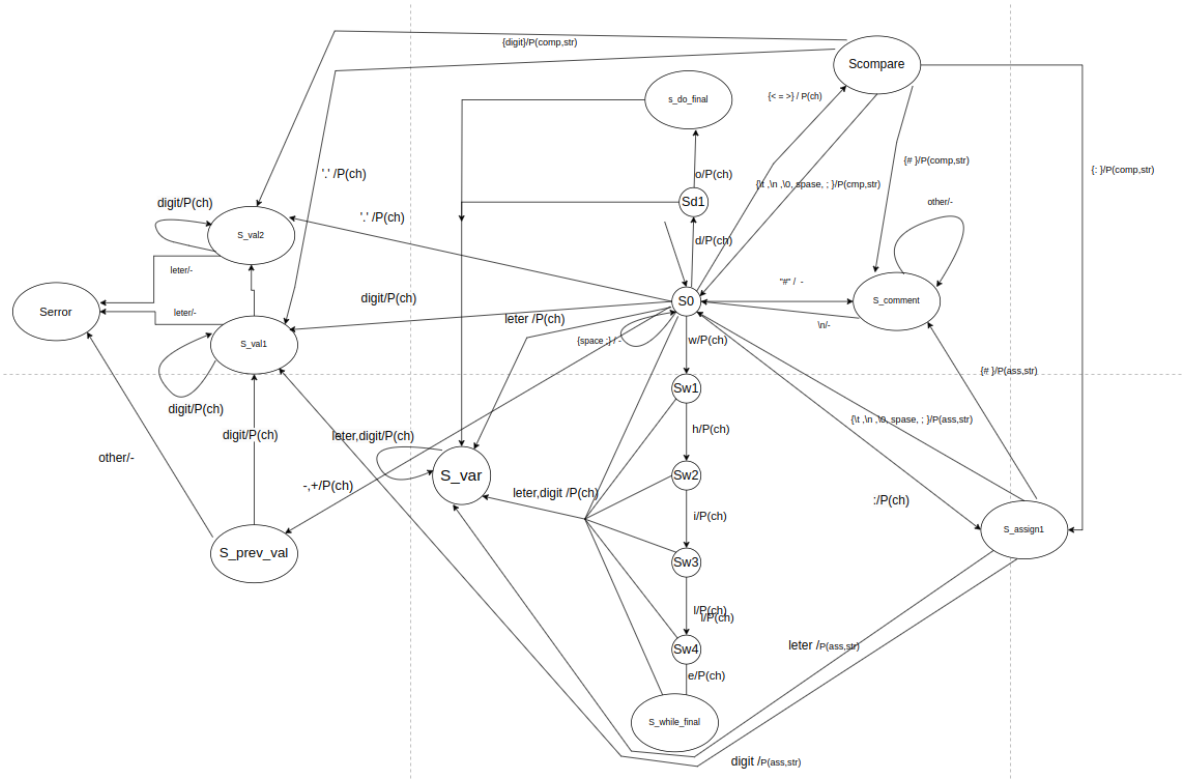

Рис. 3: Общая схема конечного автомата

На графе other - символы ASCII, которые не могут входить в лексемы.
Каждое из состояний(кроме Serror и Scomment) представлено в виде отдель-

ного автомата, чтобы не перегружать граф.

Состояния $Sd1, S\_do\_final, Sw1, Sw2, Sw3, Sw4, S\_while\_final, S\_val1, Sval2$ - также представлены в виде конечного автомата (рис. 4).

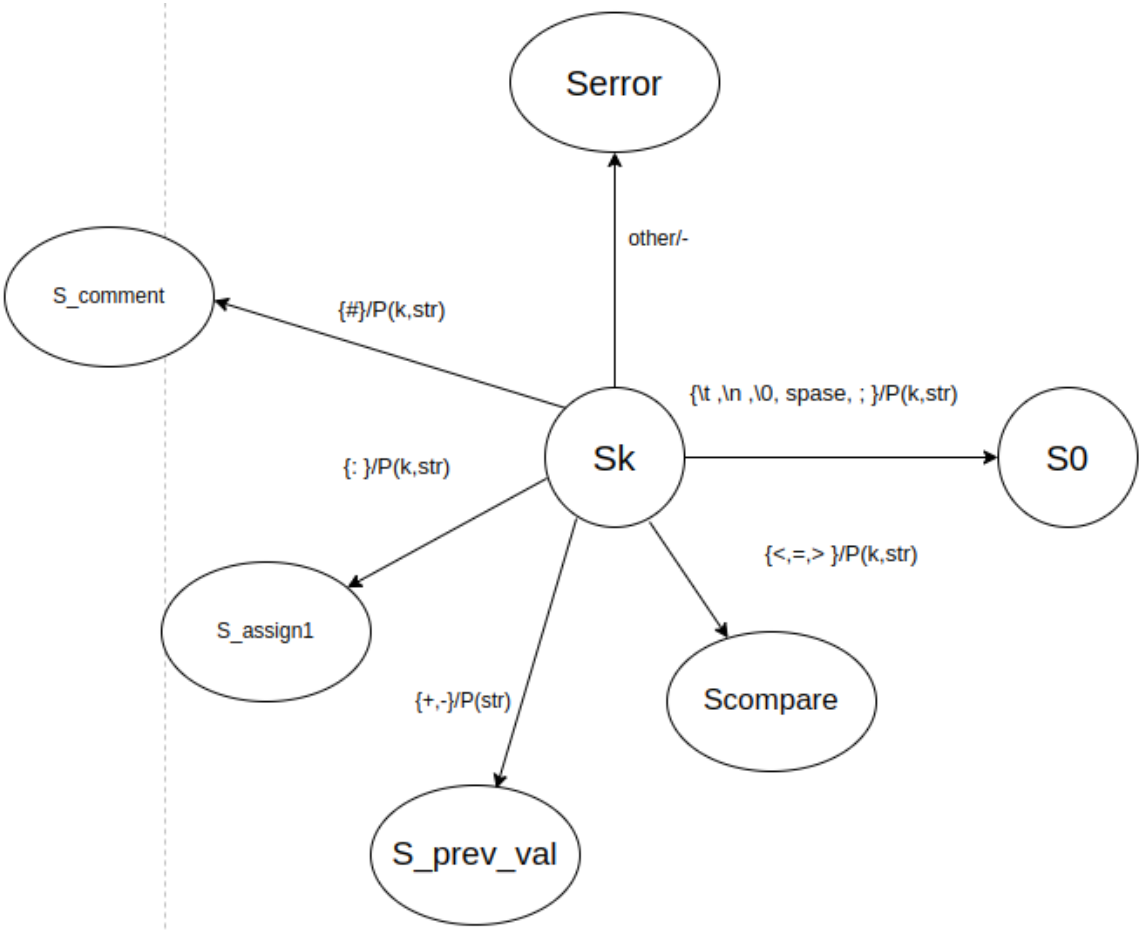

Рис. 4: Элемент декомпозиции №1

Состояния $S0, S\_assign, S\_compare$ - также представлены в виде конечного автомата (рис. 5).

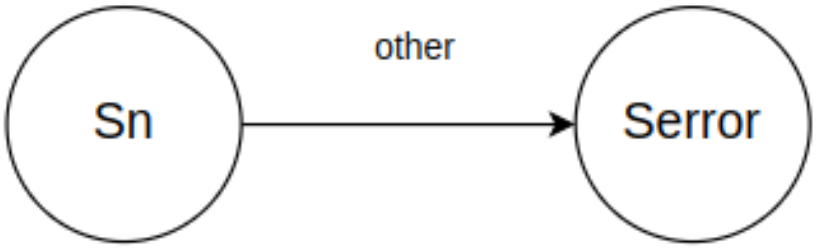

Рис. 5: Элемент декомпозиции №2

2 Особенности реализации

В рамках данной лабораторной был реализован класс LexicalAnalyzer, который содержит содержит блок перечисления состояний автомата, а так же следующие основные методы:

void CreateTable();

vector<pair<string,string> > Analyse(string);

pair<string,string> GenerateOutput();

В функции main происходит работа с файлами ввода и вывода, вызовы методов класса LexicalAnalyzer.

2.1 Реализация состояний конечного автомата

Состояния автомата описанны в виде блока перечисления:

```
enum State{
 S0,                 //0
 S_DO1,              //1
 S_DO_FINAL,         //2
 S_W1,               //3
 S_W2,               //4
 S_W3,               //5
 S_W4,               //6
 S_WHILE_FINAL,//7
 S_VARIABLE,              //8
 S_INTEGER,          //9
 S_FLOAT_VAL,             //10
 S_COMMENT,          //11

 S_ASSIGN,           //12
 S_ERROR,        //13
 S_COMPARE,  //14
 S_SEMICOLON,//15
 S_PREVINT//16
};
```

2.2 Реализация таблицы переходов конечного автомата

Таблица переходов реализована в методе CreateTable() в виде матрицы.
Программная реализация соответсвует таблице представленной на рис. 1 - 2
Исходный код функции не приведен, потому что он громоздок и тривиален.

2.3 Реализация метода Analyse(string)

Входные данные: Строка, представляющая исходный код анализируемой программы.

Выходные данные: Таблица лексем и их значений в виде
std: vector<pair<string,string> >

В теле функции при помощи цикла реализован проход по каждому из символов входной строки.

На каждой итерации:

В зависимости от входного символа происходит обновление текущего состояния на основе таблицы переходов;

Вызывается метод GenerateOutput()(Описан ниже);

Очищается строка значения текущей лексемы, текущая лексема распознана и добавлена в таблицу.

Выводится сообщение об ошибке в случае некорректной лексемы.

Исходный код функции Analyse(string str):

```
vector<pair<string,string>> LexicalAnalyzer::Analyse(string str){

    vector<pair<string,string>>  ans;
    for(int elem:str){

        pair<string,string> temp_pair= GenerateOutput();
        if(temp_pair.first!="comment" &&
         temp_pair.first!="" &&
         temp_pair.second!=""){
            ans.push_back(temp_pair);
            curr_str.clear();
        }
        if(elem!='\n' && elem!='\t' &&
         elem!='\0' && elem!=' '&& elem!=';'){
            curr_str.push_back(elem);
        }

        prev_state=curr_state;
        curr_state=table_state[curr_state][elem];
        if(curr_state==S_ERROR ||curr_str.size()>16) {
            curr_state=S0;
            prev_state=S0;
            cout<< "Error in string:  "<< curr_str;
            curr_str.clear();
        }
    }
    if(!curr_str.empty()) {
        pair<string, string> temp_pair = GenerateOutput();
        if (temp_pair.first != "comment" && temp_pair.second != "") {
            ans.push_back(temp_pair);
```

```
        }
    }

    return ans;
};
```

2.4 Реализация метода GenerateOutput()

Входные данные: Текущее и предыдущее состояние автомата.

Выходные данные: Пара(тип лексемы, значение лексемы) в виде std: pair<string,string>

В теле функции в блоке условий на основе текущего и предыдущего состояний генерируется тип лексемы.

Исходный код функции GenerateOutput():

```
pair<string,string> LexicalAnalyzer::GenerateOutput(){

    pair<string,string> ret;
    if(curr_state==S0 || curr_state==S_ASSIGN
    ||curr_state==S_ASSIGN ||curr_state==S_COMPARE) {
        if (prev_state == S0) {
            ret = make_pair("Compare operator", curr_str);
        }
        if (prev_state == S_WHILE_FINAL || prev_state == S_DO_FINAL) {
            ret = make_pair("Key word", curr_str);
        }
        if (prev_state == S_W1
        || prev_state == S_W2 || prev_state == S_W3 ||
        prev_state == S_W4 ||
            prev_state == S_DO1
             || prev_state == S_VAR) {

            ret = make_pair("Variable", curr_str);
        }
        if (prev_state == S_FLOAT_VAL) {
            ret = make_pair("Number", curr_str);
        }
        if (prev_state == S_INTEGER) {
            ret = make_pair("Number", curr_str);
        }
        if (prev_state == S_ASSIGN) {
            ret = make_pair("Assignment operator", curr_str);
        }
        if (prev_state == S_COMMENT) {
```

```
            ret = make_pair("comment", "");
        }
    }


    return ret;

}
```

2.5 Реализация функции main

Входные данные: Строка исходного кода программы.

Выходные данные: Таблица, содержащая тип лексемы, значение лексемы, номер лексемы.

В теле функции:

Читается входная строка из файла,

Вызывается метод Analyse(str) объекта класса LexicalAnalyzer,

Для каждой лексемы генерируется ее номер с помощью функции FromPairToTr(vp).

Содержимое таблицы выводится в файл markdown.

В теле функции FromPairToTr происходит подсчет номера каждой лексемы путем прохода по последовательности вложеным циклом for.

Исходный код функций main и FromPairToTr(vector<pair<string,string> > vp):

```
int main() {
    LexicalAnalyzer LA;
    string test_str=ReadFile("input.txt");
    vector<pair<string,string>>  vp=LA.Analyse(test_str);
    vector<pair<pair<string,string>,int>> out=
     FromPairToTr(vp);
    OutputToMarkdown(out, "output.md");
    return 0;
}
```

```
vector<pair<pair<string,string>,int>>
 FromPairToTr(vector<pair<string,string>> vp){
    vector<pair<string,string>> ret;
    vector<pair<pair<string,string>,int>> vtr;
```

```
    for(int i=0;i<vp.size();i++){
        int counter=1;
        for(int j=0;j<i;j++){
            if(vp[i].first==vp[j].first &&vp[i].second!=vp[j].second )
                counter++;
        }
        vtr.push_back(make_pair(vp[i],counter));

    }
    return vtr;
}
```

3 Результат работы

3.1 Пример обработки корректной строки

В качестве входного примера была взята следующая строка исходного кода:

```
a:=100
b:=1000
a1:=1100
b1:=20
while a<a1 do
    a increase

do a:=.1 b:= 67.9 #something
while a>b b<a;
```

. Результат работы программы можно увидеть на рис. 6.

Лексема	Тип лексемы	Значение
a	Variable	a : 1
:=	Assignment operator	
100	Number	100
b	Variable	b : 2
:=	Assignment operator	
1000	Number	1000
a1	Variable	a1 : 3
:=	Assignment operator	
1100	Number	1100
b1	Variable	b1 : 4
:=	Assignment operator	
20	Number	20
while	Key word	X1
a	Variable	a : 1
<	Compare operator	cmp : <
a1	Variable	a1 : 3
do	Key word	X2
a	Variable	a : 1
increase	Variable	increase : 5
do	Key word	X2
a	Variable	a : 1
:=	Assignment operator	
.1	Number	.1
b	Variable	b : 2
:=	Assignment operator	
67.9	Number	67.9
while	Key word	X1
a	Variable	a : 1
>	Compare operator	cmp : >
b	Variable	b : 2
b	Variable	b : 2
<	Compare operator	cmp : <
a	Variable	a : 1
;	Semicolon	

Рис. 6: Таблица лексем №1

3.2 Пример обработки строки с ошибками

В качестве входного примера была взята следующая строка исходного кода:

```
b=2.2.
a=10
q!=20
945ds
while
whil
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
#aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

Программа выведет сообщение об ошибке в строке рис. 7 и составит следующую таблицу лексем: рис. 8:

```
Error in string:   "2.2."
Unknown simbol:    !
Error in string:   "945ds"
Error in string:   "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"
```

Рис. 7: Сообщение об ошибке

7.

Лексема	Тип лексемы	Значение
b	Variable	b : 1
=	Compare operator	cmp : =
a	Variable	a : 2
=	Compare operator	cmp : =
10	Number	10
q	Variable	q : 3
=	Compare operator	cmp : =
20	Number	20
while	Key word	X1
whil	Variable	whil : 4

Рис. 8: Таблица лексем №2

Заключение

Итак, в результате работы была реализована программа, представляющая собой лексический анализатор входной последовательности символов в соответствии с указаным вариантом(14).

Достоинства реализации:

1. Реализация переходов автомата с помощью таблицы(а не блока условий), что позволяет легче масштабировать программу.

2. Вывод таблицы в файл markdown.

Недостатки реализации:

1. Реализация функции выходов автомата с помощью блока условий.

2. Квадратичная алгоритмическая сложность функции FromPairToTr.

Масштабирование:

Программу можно масштабировать путем добаления новых ключевых слов. Для этого нужно только изменить таблицу переходов в методе CreateTable().

Кроме этого, программу можно масштабировать путем добавления новых типов лексем, однако для этого уже придется изменять не только таблицу, но и функцию выходов.

Список использованных источников

[1] Ф.А Новиков. Дискретная математика для программистов: Учебник для вузов. 3-е изд. — СПб: Питер, 2008. —384 с.

[2] А.В. Востров. Лекция №2 по дисциплине «Математическая логика и теория автоматов». 2024г. 42 сл.