Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Кафедра информатики
кафедра

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

Разработка лексического анализа компилятора
тема

Преподаватель _____ А. С. Кузнецов
подпись, дата — инициалы, фамилия

Студент КИ19-04-1М, 031943329 _____ И. С. Байкалов
номер группы, зачетной книжки — подпись, дата — инициалы, фамилия

Красноярск 2020

СОДЕРЖАНИЕ

1 Цель

Изучить методы лексического анализа и их программную реализацию.

2 Ход работы

Файл flex делится на 3 части, а именно:

- объявление переменных, констант;

- правила;

- вспомогательные функции.

В ходе работы была реализован код, представленный в таблице 1.

Таблица 1 – листинг кода tokens.l

```
%{
    #include <stdio.h>
    int chars = 0;
%}

%option yylineno
CHAR     [^"\n]
DIGIT    [0-9]
%%

\n                              { chars = 0; }

[ \t\r]                         { chars++; }

{DIGIT}+"."{DIGIT}*            { printf("%s\t\t->\t FLOAT\t(%d:%d)\n",
yytext, yylineno, ++chars); chars += strlen(yytext) - 1; }
{DIGIT}+                        { printf("%s\t\t->\t
INTEGER\t(%d:%d)\n", yytext, yylineno, ++chars); chars +=
strlen(yytext) - 1; }
\'{CHAR}{1}\'                   { printf("%s\t\t->\t CHAR\t\t(%d:%d)\n",
yytext, yylineno, ++chars); chars += strlen(yytext) - 1; }
\"{CHAR}*\"                     { printf("%s\t\t->\t
STRING\t\t(%d:%d)\n", yytext, yylineno, ++chars); chars +=
strlen(yytext) - 1; }

integer|float|string|char       { printf("%s\t\t->\t DATA
TYPE\t(%d:%d)\n", yytext, yylineno, ++chars); chars += strlen(yytext) -
1; }
if|else|elif|while              { printf("%s\t\t->\t
KEYWORD\t(%d:%d)\n", yytext, yylineno, ++chars); chars +=
strlen(yytext) - 1; }

[a-zA-Z][a-zA-Z0-9]*           { printf("%s\t\t->\t
```

```
IDENTIFIER\t(%d:%d)\n", yytext, yylineno, ++chars); chars +=
strlen(yytext) - 1; }

\+                      { printf("%s\t\t->\t PLUS\t\t(%d:%d)\n",
yytext, yylineno, ++chars); chars += strlen(yytext) - 1; }
\-                      { printf("%s\t\t->\t MINUS\t\t(%d:%d)\n",
yytext, yylineno, ++chars); chars += strlen(yytext) - 1; }
\/                      { printf("%s\t\t->\t DIVISION\t\t(%d:%d)\n",
yytext, yylineno, ++chars); chars += strlen(yytext) - 1; }
\*                      { printf("%s\t\t->\t MULTIPLY\t\t(%d:%d)\n",
yytext, yylineno, ++chars); chars += strlen(yytext) - 1; }
\=                      { printf("%s\t\t->\t ASSIGN\t\t(%d:%d)\n",
yytext, yylineno, ++chars); chars += strlen(yytext) - 1; }
\=\=                     { printf("%s\t\t->\t EQUALS\t\t(%d:%d)\n",
yytext, yylineno, ++chars); chars += strlen(yytext) - 1; }
\<                      { printf("%s\t\t->\t LESS\t\t(%d:%d)\n",
yytext, yylineno, ++chars); chars += strlen(yytext) - 1; }
\>                      { printf("%s\t\t->\t MORE\t\t(%d:%d)\n",
yytext, yylineno, ++chars); chars += strlen(yytext) - 1; }

\(                      { printf("%s\t\t->\t OPENPAREN\t(%d:%d)\n",
yytext, yylineno, ++chars); chars += strlen(yytext) - 1; }
\)                      { printf("%s\t\t->\t CLOSEPAREN\t(%d:%d)\n",
yytext, yylineno, ++chars); chars += strlen(yytext) - 1; }
\{                      { printf("%s\t\t->\t OPENBRACE\t(%d:%d)\n",
yytext, yylineno, ++chars); chars += strlen(yytext) - 1; }
\}                      { printf("%s\t\t->\t CLOSEBRACE\t(%d:%d)\n",
yytext, yylineno, ++chars); chars += strlen(yytext) - 1; }

\.                      { printf("%s\t\t->\t DOT\t(%d:%d)\n", yytext,
yylineno, ++chars); chars += strlen(yytext) - 1; }
\,                      { printf("%s\t\t->\t COMMA\t(%d:%d)\n", yytext,
yylineno, ++chars); chars += strlen(yytext) - 1; }
\:                      { printf("%s\t\t->\t COLON\t(%d:%d)\n", yytext,
yylineno, ++chars); chars += strlen(yytext) - 1; }
\;                      { printf("%s\t\t->\t SEMICOLON\t(%d:%d)\n",
yytext, yylineno, ++chars); chars += strlen(yytext) - 1; }

\/\/\S*[a-zA-Z][a-zA-Z0-9]*    { printf("%s\t->\t COMMENT\t(%d:%d)\n",
yytext, yylineno, ++chars); chars += strlen(yytext) - 1; }

\"{CHAR}*$                      { printf("%s\t\t->\t STRING
ERROR\t\t(%d:%d)\n", yytext, yylineno, ++chars); chars +=
strlen(yytext) - 1; }
.                               { chars++; printf("Error in line
%d:%d\n", yylineno, chars); exit(1); }

%%

int yywrap(){ return 0; }

int main(int argc, const char* argv[]) {

    char filename [255];

    FILE*file;
```

```
    do {
        printf ("path to file: ");
        scanf ("%s", filename);
        file = fopen(filename, "r");
        if (file == NULL) {
            printf ("Can't open file \n");
        }
    } while(file == NULL);

    yyin = file;

    yylex();
    fclose(file);
    printf("\nOlmeca is good\n");
}
```

Далее, после написания кода на flex, исходный код нужно пропустить через компилятор flex командой flex tokens.l и получить файл lex.yy.c.

После этого файл lex.yy.c. нужно пропустить через компилятор C командой cc lex.yy.c и получить файл a.out, который уже можно запустить в терминале.

Для тестирования использовались 3 файла 1.olm, 2.olm и 3.olm.

На рисунках 4 - 6 можно увидеть результат выполнения работы.

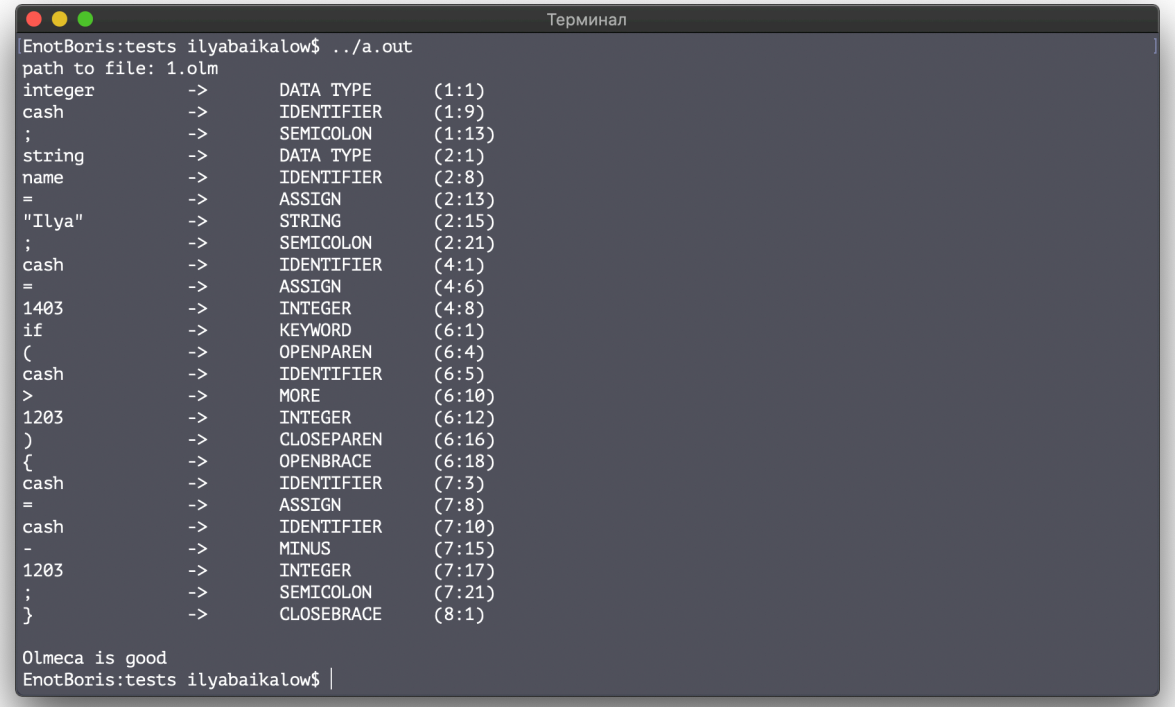

Рисунок 4 – Таблица токенов для 1.olm

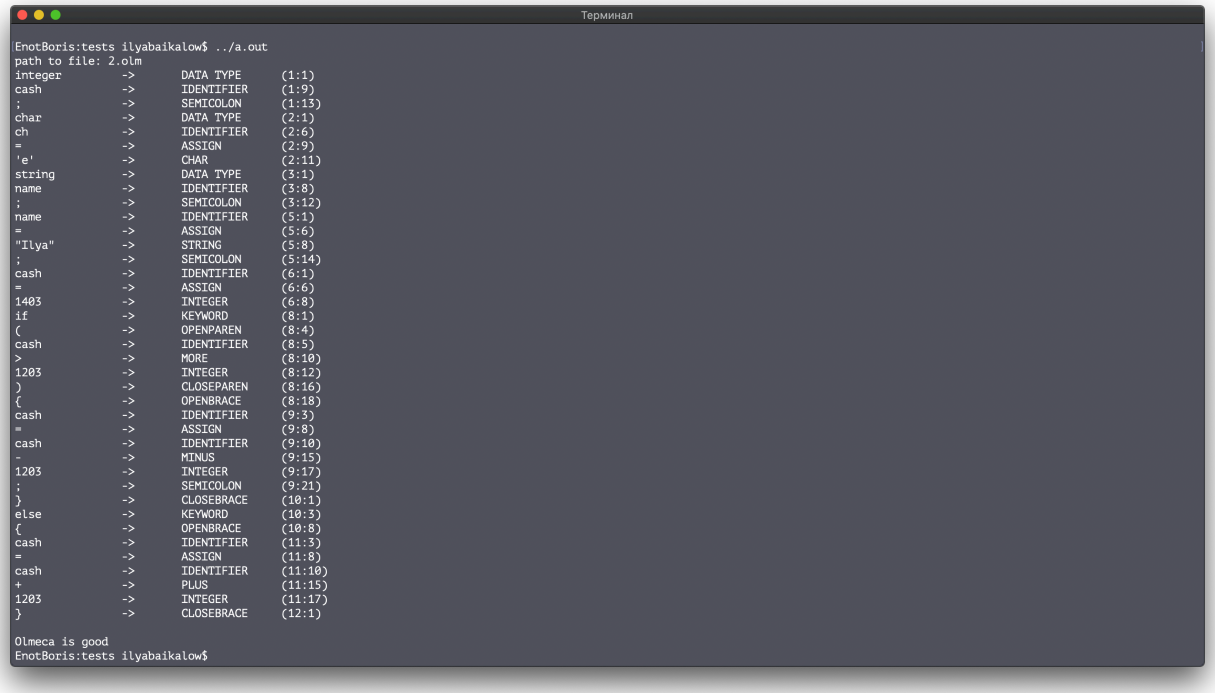

Рисунок 5 – Таблица токенов для 2.olm

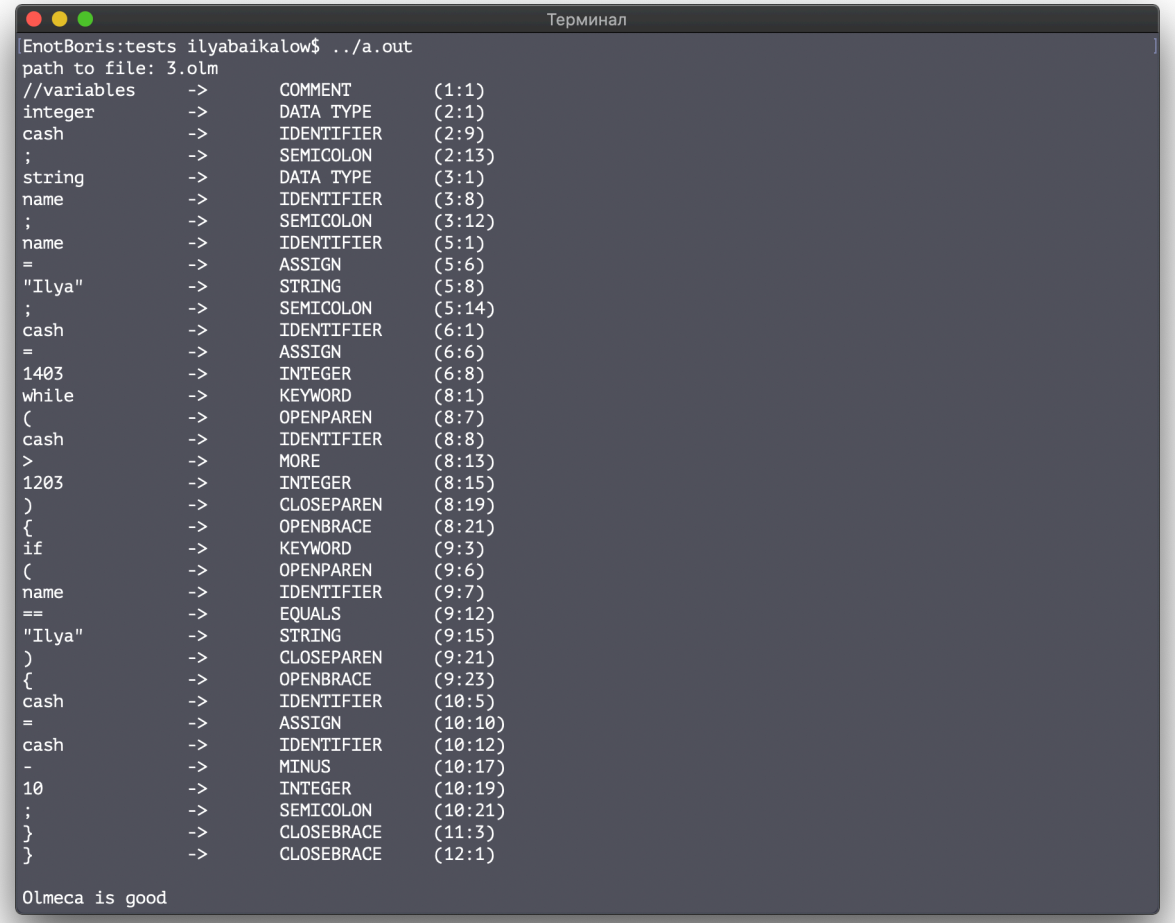

Рисунок 6 – Таблица токенов для 3.olm

3 Вывод

В ходе лабораторной работы бел реализован лексический анализатор с использованием flex. Анализатор выводит в консоль таблицу токенов, также показывает номер строки и позицию в строке.