МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра «Вычислительные системы и технологии»

Проектирование лексического анализатора

ОТЧЁТ

по лабораторной работе №1

по дисциплине

Теория языков программирования и методы трансляции

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кузнецов Г.Д.

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Малинок С.М.

21-ПО

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

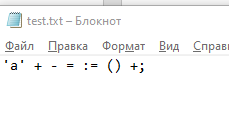
С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

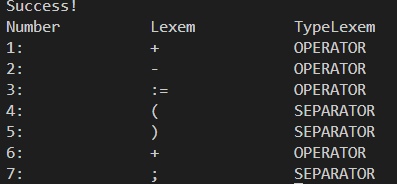
Нижний Новгород 2024

Задание:

Написать программу, которая выполняет лексический анализ входного текста в соответствии с заданием и порождает таблицу лексем с указанием их типов и значение. Входной язык содержит арифметические выражения, разделенные символами ; + - \* / ( ) :=.

Результат работы:

****



**Схема Бэкуса-Наура:**

S 🡪 B | B S

B 🡪 for (W; W; W) do | for (W; W; W) do (B)

W 🡪 I G I | I G N | N G N | I := I | I := W | I A I | (W)

I 🡪 L | L I | \_ | \_ I | I L I

A 🡪 + | - | \* | / | %

G 🡪 > | < | == | !=

L 🡪 a | b | … | z | A | B | … | Z

R 🡪 N | RN

S – стартовый терминал

W – выражение

I – идентификатор

A – арифметические знаки

G – знаки сравнения

L – все буквы алфавита

R - числа

**Код программы:**

// ТяпЛяп.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

//

#include <stdio.h>

#include <iostream>

#include <string>

#include <vector>

using namespace std;

const string keywords[] = { "for", "do", "int" };

enum state\_type { START, CONST, IDEN, NUM, ASSIGN, SPR, ARIFM, COM, CONDITIONS, ER, SYM };

enum lexem\_type { KEYWORD, CONSTANT, IDENTIFICATOR, NUMBER, OPERATOR, SEPARATOR, ERROR, SYMBOL };

struct lexem

{

    lexem\_type name;

    string value;

};

int lexAnaliz(const char\* filename, vector<lexem>\* table\_lexem);

bool is\_keyword(char\* lexem);

string get\_string\_type(lexem\_type name);

int main()

{

    vector<lexem> table\_lexem;

    const char\* filename = "test.txt";

    if (lexAnaliz(filename, &table\_lexem) == 0)

        std::cout << "Success!" << std::endl;

    else

        std::cout << "Something wrong..." << std::endl;

    std::cout << "Number\t\tLexem\t\tTypeLexem" << std::endl;

    ;

    for (size\_t i = 0; i < table\_lexem.size(); i++)

    {

        std::cout << (i + 1) << ":" "\t\t" << table\_lexem[i].value << "\t\t" << get\_string\_type(table\_lexem[i].name) << std::endl;;

    }

    return 1;

}

int lexAnaliz(const char\* filename, vector<lexem>\* table\_lexem)

{

    // Для вывода сообщения, о расположение ошибки

    int number\_line = 1;

    int number\_symbol = 0;

    // Переменные, для того, чтобы корректно обрабатывать лексемы

    // начинающие с римских цифр, являющиемися идентификаторами

    bool flag\_id\_with\_number = false;

    string buff\_id\_with\_number = "";

    // исходный файл

    FILE\* file;

    // Текущий обрабатываемый символ

    char cur\_sym;

    // текущая лексема

    lexem cur\_lexem;

    fopen\_s(&file, filename, "r");

    if (file == NULL)

    {

        std::cout << "Error open file!" << std::endl;

        return -1;

    }

    state\_type state = START;

    cur\_sym = fgetc(file);

    number\_symbol++;

    while (!feof(file))

    {

        switch (state)

        {

            // Начальное состоние. Начинаем обрабатывать новую лексему

        case START:

            // Пропускаем символы пробел, табуляцию и переход

            // на следующую строчку

            while ((cur\_sym == ' ') || (cur\_sym == '\t') || (cur\_sym == '\n'))

            {

                if (cur\_sym == '\n')

                {

                    number\_line++;

                    number\_symbol = 0;

                }

                cur\_sym = fgetc(file);

                number\_symbol++;

            }

            // Если это X V I то переходим в состояние NUM

            if (cur\_sym == 'X' || cur\_sym == 'V' || cur\_sym == 'I')

            {

                state = NUM;

            }

            else if (cur\_sym == '"' )

            {

                state = CONST;

            }

            else if (cur\_sym == '\'')

            {

                state = SYM;

            }

            // Если первый символ буква или \_, то переходим в состояния IDEN

            else if (

                ((

                    (cur\_sym >= 'A') && (cur\_sym <= 'Z')

                    )

                    ||

                    (

                        (cur\_sym >= 'a') && (cur\_sym <= 'z')

                        )

                    ||

                    (cur\_sym == '\_'))

                )

            {

                state = IDEN;

            }

            else if (cur\_sym == '+' || cur\_sym == '-' || cur\_sym == '\*' ||

                cur\_sym == '/' || cur\_sym == '%')

            {

                state = ARIFM;

            }

            else if (cur\_sym == ':')

            {

                state = ASSIGN;

            }

            else if (cur\_sym == '(' || cur\_sym == ')' || cur\_sym == ';' ||

                cur\_sym == '{' || cur\_sym == '}')

            {

                state = SPR;

            }

            else if (cur\_sym == '<' || cur\_sym == '>' || cur\_sym == '=' ||

                cur\_sym == '!')

            {

                state = CONDITIONS;

            }

            // Если не одна проверка не прошла, то переходим в состояние ERROR

            else

            {

                state = ER;

            }

            break;

            // Состояние индентификатор - возможно, это имя переменой или ключевое слово.

        case IDEN:

            int size\_iden;

            size\_iden = 0;

            char lexem\_iden[32];

            lexem\_iden[size\_iden] = cur\_sym;

            size\_iden++;

            cur\_sym = fgetc(file);

            number\_symbol++;

            // цикл, который будет заполнять lexem\_id до тех пор, пока

            // встречаются символы алфавита или \_

            while (

                ((cur\_sym >= 'A') && (cur\_sym <= 'Z')) ||

                ((cur\_sym >= 'a') && (cur\_sym <= 'z')) ||

                (cur\_sym == '\_'))

            {

                if (size\_iden > 31) break;

                lexem\_iden[size\_iden] = cur\_sym;

                size\_iden++;

                cur\_sym = fgetc(file);

                number\_symbol++;

            }

            if (size\_iden > 31) break;

            lexem\_iden[size\_iden] = '\0';

            // Если это Ключевое слово, то помечаем имя токена Keyword, иначе Identificator

            if (is\_keyword(lexem\_iden))

                cur\_lexem.name = KEYWORD;

            else

                cur\_lexem.name = IDENTIFICATOR;

            // Идентификатор начинающийся с римским символов

            if (flag\_id\_with\_number == true)

            {

                cur\_lexem.value = buff\_id\_with\_number;

                cur\_lexem.value += lexem\_iden;

                flag\_id\_with\_number = false;

                buff\_id\_with\_number = "";

            }

            else

            {

                cur\_lexem.value = lexem\_iden;

            }

            (\*table\_lexem).push\_back(cur\_lexem);

            state = START;

            break;

            // Состояние числа - возможно что следующая лексема это римское число

        case NUM:

            int size\_num;

            size\_num = 0;

            char lexem\_num[32];

            lexem\_num[size\_num] = cur\_sym;

            size\_num++;

            cur\_sym = fgetc(file);

            number\_symbol++;

            // цикл, который будет заполнять lexem\_id до тех пор, пока

            // встречаются символы X V I

            while (cur\_sym == 'X' || cur\_sym == 'V' || cur\_sym == 'I')

            {

                if (size\_num > 31) break;

                lexem\_num[size\_num] = cur\_sym;

                size\_num++;

                cur\_sym = fgetc(file);

                number\_symbol++;

            }

            // не знаю как по красивее это сделать

            if (size\_num > 31) break;

            lexem\_num[size\_num] = '\0';

            cur\_lexem.name = NUMBER;

            cur\_lexem.value = lexem\_num;

            // Если следующий символ после римских цифр стоит буква алфавита - то это identificator

            if (

                ((

                    (cur\_sym >= 'A') && (cur\_sym <= 'Z')

                    )

                    ||

                    (

                        (cur\_sym >= 'a') && (cur\_sym <= 'z')

                        )

                    ||

                    (cur\_sym == '\_'))

                )

            {

                state = IDEN;

                flag\_id\_with\_number = true;

                buff\_id\_with\_number = lexem\_num;

                break;

            }

            (\*table\_lexem).push\_back(cur\_lexem);

            state = START;

            break;

        case CONST:

            int size\_const;

            size\_const = 0;

            char lexem\_const[32];

            lexem\_const[size\_const] = cur\_sym;

            size\_const++;

            cur\_sym = fgetc(file);

            number\_symbol++;

            // цикл, который будет заполнять lexem\_id до тех пор, пока

            // встречаются символы встречаются символы

            //while ((cur\_sym != ' ') && (cur\_sym != '\t') && (cur\_sym != '\n') && (cur\_sym != '\0'))

            //while (cur\_sym != '"')

            while (true)

            {

                if (size\_const > 31) break;

                lexem\_const[size\_const] = cur\_sym;

                size\_const++;

                cur\_sym = fgetc(file);

                number\_symbol++;

                if (cur\_sym == '"') {

                    lexem\_const[size\_const] = cur\_sym;

                    size\_const++;

                    break;

                }

            }

            // не знаю как по красивее это сделать

            if (size\_const > 31) break;

            lexem\_const[size\_const] = '\0';

            cur\_lexem.name = CONSTANT;

            cur\_lexem.value = lexem\_const;

            (\*table\_lexem).push\_back(cur\_lexem);

            cur\_sym = fgetc(file);

            number\_symbol++;

            state = START;

            break;

            // состояние приравнивание - возможно это знак :=

        case ASSIGN:

            cur\_sym = fgetc(file);

            number\_symbol++;

            if (cur\_sym == '=')

            {

                cur\_lexem.name = OPERATOR;

                cur\_lexem.value = ":=";

                (\*table\_lexem).push\_back(cur\_lexem);

                cur\_sym = fgetc(file);

                number\_symbol++;

                state = START;

            }

            else

                state = ER;

            break;

            // состояние разделителей - это один из символов (){};

        case SPR:

            cur\_lexem.name = SEPARATOR;

            cur\_lexem.value = cur\_sym;

            (\*table\_lexem).push\_back(cur\_lexem);

            cur\_sym = fgetc(file);

            number\_symbol++;

            state = START;

            break;

        case SYM:

            char noflag;

            noflag = cur\_sym;

            number\_symbol++;

            cur\_sym = fgetc(file);

            char flag;

            flag = cur\_sym;

            number\_symbol++;

            cur\_sym = fgetc(file);

            if (cur\_sym == '\'')

            {

                cur\_lexem.name = SYMBOL;

                cur\_lexem.value = flag;

                number\_symbol++;

                cur\_sym = fgetc(file);

                (\*table\_lexem).push\_back(cur\_lexem);

                state = START;

            }

            else {

                if (((noflag >= 'A') && (noflag <= 'Z')) ||

                    ((noflag >= 'a') && (noflag <= 'z'))) {

                    cur\_lexem.name = IDENTIFICATOR;

                    cur\_lexem.value = noflag;

                    (\*table\_lexem).push\_back(cur\_lexem);

                    state = START;

                }

                else if (noflag == ' ') {

                    state = START;

                }

                else {

                    cur\_lexem.name = ERROR;

                    cur\_lexem.value = noflag;

                    (\*table\_lexem).push\_back(cur\_lexem);

                }

                if (((flag >= 'A') && (flag <= 'Z')) ||

                    ((flag >= 'a') && (flag <= 'z')) ) {

                    cur\_lexem.name = IDENTIFICATOR;

                    cur\_lexem.value = flag;

                    (\*table\_lexem).push\_back(cur\_lexem);

                    state = START;

                }

                else if (flag == ' ') {

                    state = START;

                }

                else {

                    cur\_lexem.name = ERROR;

                    cur\_lexem.value = flag;

                    (\*table\_lexem).push\_back(cur\_lexem);

                }

                //number\_symbol++;

                //cur\_sym = fgetc(file);

                state = START;

            }

            break;

        case CONDITIONS:

            cur\_lexem.name = OPERATOR;

            cur\_lexem.value = cur\_sym;

            cur\_sym = fgetc(file);

            number\_symbol++;

            (\*table\_lexem).push\_back(cur\_lexem);

            state = START;

            /\*if (cur\_sym == '=' || cur\_sym == '!')

            {

                cur\_lexem.name = OPERATOR;

                cur\_lexem.value = cur\_sym;

                cur\_sym = fgetc(file);

                number\_symbol++;

                if (cur\_sym != '=')

                {

                    state = ER;

                    break;

                }

                (\*table\_lexem).push\_back(cur\_lexem);

            }

            else if ((cur\_sym == '<') || (cur\_sym == '>'))

            {

                cur\_lexem.name = OPERATOR;

                cur\_lexem.value = cur\_sym;

                cur\_sym = fgetc(file);

                number\_symbol++;

                (\*table\_lexem).push\_back(cur\_lexem);

                state = START;

            }\*/

            break;

        case ARIFM:

            cur\_lexem.name = OPERATOR;

            cur\_lexem.value = cur\_sym;

            cur\_sym = fgetc(file);

            number\_symbol++;

            if (cur\_sym == '\*' || cur\_sym == '/')

            {

                state = COM;

                break;

            }

            (\*table\_lexem).push\_back(cur\_lexem);

            state = START;

            break;

        case COM:

            // Большой комментарий

            if (cur\_sym == '\*')

            {

                cur\_sym = fgetc(file);

                while (true)

                {

                    while (cur\_sym != '\*')

                    {

                        if (cur\_sym == '\n')

                        {

                            number\_line++;

                            number\_symbol = 0;

                        }

                        cur\_sym = fgetc(file);

                        number\_symbol++;

                    }

                    cur\_sym = fgetc(file);

                    number\_symbol++;

                    if (cur\_sym == '\n')

                    {

                        number\_line++;

                        number\_symbol = 0;

                    }

                    if (cur\_sym == '/') break;

                    cur\_sym = fgetc(file);

                    number\_symbol++;

                }

            }

            // Маленький комментарий

            else

            {

                while (cur\_sym != '\n' && !feof(file))

                {

                    cur\_sym = fgetc(file);

                    number\_symbol++;

                }

                number\_line++;

                number\_symbol = 0;

            }

            state = START;

            cur\_sym = fgetc(file);

            number\_symbol++;

            break;

            // состояние ошибки - не понятный символ, или какая-то ошибка

        case ER:

            std::cout << "Error in the line:" << number\_line << "   number symbol:" << number\_symbol << std::endl;

            /\*while ((cur\_sym != ' ') && (cur\_sym != '\t') && (cur\_sym != '\n'))

            {

                if (cur\_sym == '\n')

                {

                    number\_line++;

                    number\_symbol = 0;

                }

                cur\_sym = fgetc(file);

                number\_symbol++;

                state = START;

            }\*/

            cur\_lexem.name = ERROR;

            cur\_lexem.value = cur\_sym;

            (\*table\_lexem).push\_back(cur\_lexem);

            number\_symbol++;

            cur\_sym = fgetc(file);

            state = START;

            break;

        default:

            break;

        }

    }

    if (file != NULL)

    {

        fclose(file);

    }

    return 0;

}

bool is\_keyword(char\* lexem)

{

    for (size\_t i = 0; i < keywords->size(); i++)

    {

        if (keywords[i] == lexem) return true;

    }

    return false;

}

string get\_string\_type(lexem\_type name)

{

    switch (name)

    {

    case KEYWORD:

        return "KEYWORD";

    case IDENTIFICATOR:

        return "IDENTIFICATOR";

    case NUMBER:

        return "NUMBER";

    case OPERATOR:

        return "OPERATOR";

    case SEPARATOR:

        return "SEPARATOR";

    case CONSTANT:

        return "CONSTANT";

    case ERROR:

        return "ERROR";

    case SYMBOL:

        return "SYMBOL";

    default:

        return "default";

    }

}

Вывод:

В ходе выполнения лабораторной работы, были получены навыки проектирования лексического анализатора и навыки написания схемы Бэкуса-Наура.