*Федеральное государственное бюджетное образовательное учреждение* *высшего образования*

|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | ***«Московский государственный технический университет  имени Н.Э. Баумана***  ***(национальный исследовательский университет)»***  ***(МГТУ им. Н.Э. Баумана)*** |

ФАКУЛЬТЕТ \_ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Отчет**

**по домашнему заданию № \_2\_\_\_\_**

**Дисциплина: \_**Машинно-зависимые языки и основы компиляции**\_\_\_\_**

**Название лабораторной работы: \_**Лексические и синтаксические\_\_

\_анализаторы**\_**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент гр. **\_**ИУ6-42б**\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_**И.С. Марчук**\_\_\_**

(Подпись, дата) (И.О. Фамилия)

Преподаватель  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О. Фамилия)

Москва, 2021

**Задание:**

Разработать программу на языке ассемблера, которая решает поставленную задачу.

**Вариант 16.** C++. Разработать грамматику и распознаватель прототипов функции.

Считать, что параметры только стандартных (скалярных) типов. Например: **int as3(int a\_34, float bjt, unsigned char car);**

**Цель работы:**

Закрепление знаний теоретических основ и основных методов приемов разработки лексических и синтаксических анализаторов регулярных и контекстно свободных формальных языков.

**Описание грамматики в форме Бэкуса–Наура, с указанием типа грамматики:**

<прототип>::= void(); | void(<список\_переменных>); | <переменная>(<список\_переменных>);

<список\_переменных> ::= <список\_переменных>,<переменная>|<переменная>;

<переменная> ::= <префикс> <тип> <идентификатор> | <тип> <идентификатор>;

Терминальные лексемы:

<префикс> ::= signed | unsigned;

<тип> ::= char | short | int | long | long\_long | float | double | long\_double;

<символ> ::= A|a|B|b|С|c|D|d|E|e|F|f|G|g|H|h|I|i|J|j|K|k|L|l|M|m|N|n|O|o|P|p| Q|q|R|r|S|s|T|t|U|u|V|v|W|w|X|x|Y|y|Z|z|\_|-;

<цифра> ::= 1|2|3|4|5|6|7|8|9|0;

Нетерминальные лексемы:

<идентификатор> ::= <идентификатор><символ> | <идентификатор><цифра> | <символ>

В даннном случае мы имеем третий тип грамматики несмотря на то, что сам по себе язык C++ относится к 1 типу. Эта грамматика относится к типу выше 1 так как, нет конструкций, которые бы повлияли на другие конструкции, став для них контекстом. Все нетерминальные и терминальные символы могут свободно подставляться во всех своих вариантах независимо друг от друга. Не является эта грамматика и вторым типом, тк можно реализовать конечный автомат представленный на рисунке 1 и в таблице 1. Следовательно **она относится к 3 типу.**

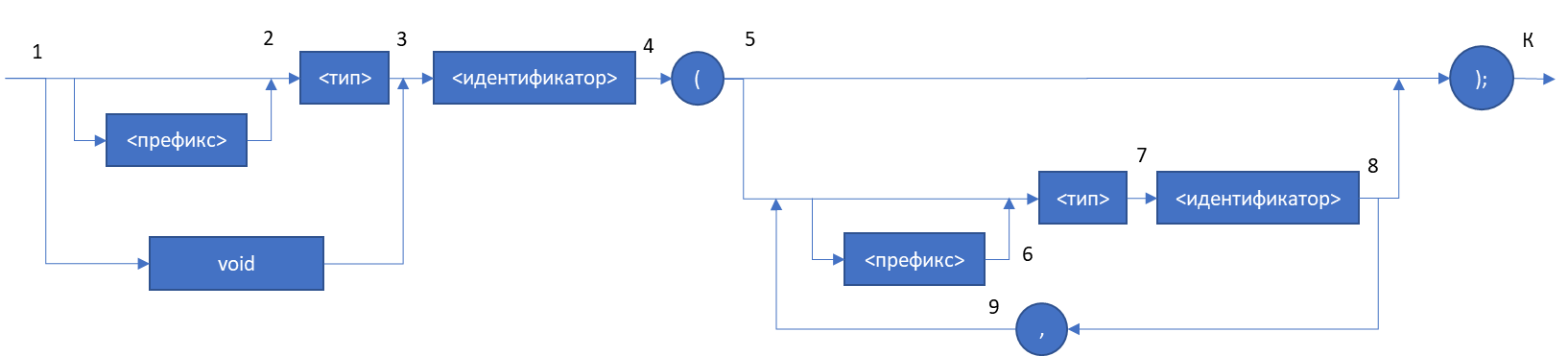


Рисунок 1 – Автомат

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Префикс | Тип | Идентиф. | void | ( | , | ); |
| 1 | 2 | 3 | E | 3 | E | E | E |
| 2 | E | 3 | E | E | E | E | E |
| 3 | E | E | 4 | E | E | E | E |
| 4 | E | E | E | E | 5 | E | E |
| 5 | 6 | 7 | E | E | E | E | K |
| 6 | E | 7 | E | E | E | E | E |
| 7 | E | E | 8 | E | E | E | E |
| 8 | E | E | E | E | E | 9 | K |
| 9 | 6 | 7 | E | E | E | E | E |

Таблица 1 – Таблица переходов автомата

**Обоснование выбора метода разбора:**

Тк на вход может поступить абсолютно случайная строка, то для разбивки строки на токены – грамматического разбора, используется восходящий метод.

После анализа грамматики получаются строки вида:

T I(T I, T I, P T I);

Где:

– “P”, это префикс;

– “T”, это тип переменной;

– “I”, это идентификатор;

– “V”, это ключевое слово void;

– “(”, это начало списка идентификаторов;

– “,”, это разделитель;

– “);”, это конец списка идентификаторов.

Для анализа синтаксиса в строках токенов был придуман конечный автомат, приведенный выше (рис. 1).

**Текст программы с комментариями:**

Функция разбора строки на токены:

static String getTokenString(String in) throws Exception {// метод разбирающий строку на токены

StringBuilder tokenString = new StringBuilder(" ");// инициализируем строку для ответа

StringBuilder inCopy = new StringBuilder(in);// копируем строку во внутреннюю переменную метода

boolean checkedFlag;// опознан ли како-нибудь символ за текущую итерацию

while (inCopy.length() != 0) {// цикл по строке

checkedFlag = false;

if (tokenString.charAt(tokenString.length() - 1) != 'I') {

// отфильтровываем строки на подобие aaachar

// ищем префиксы

String[] search = {"signed", "unsigned"};

int wordIterator = 0;

while (wordIterator < search.length && !checkedFlag) {// пробегаемся по массиву

int poz = 0;// проверяем эквивалентность цепочек

while (search[wordIterator].length() > poz && inCopy.length() > poz) {

if (search[wordIterator].charAt(poz) != inCopy.charAt(poz))

break;

poz++;

}

if (poz == search[wordIterator].length()) {// если цепочка пройдена целиком

checkedFlag = true; // символ найден

inCopy.delete(0, search[wordIterator].length()); // удаляем найденное из строки

tokenString.append('P');// добавляем токен

}

wordIterator++;// к следующему префиксу

}

{// ищем void

String s = "void";

int poz = 0;// проверяем эквивалентность цепочек

while (s.length() > poz && inCopy.length() > poz) {

if (s.charAt(poz) != inCopy.charAt(poz))

break;

poz++;

}

if (poz == s.length()) {// если цепочка пройдена целиком

checkedFlag = true; // символ найден

inCopy.delete(0, s.length()); // удаляем найденное из строки

tokenString.append('V');// добавляем токен

}

}

// ищем типы переменных

search = new String[]{

"char",

"short",

"int",

"long\_long",

"long\_double",

"long",

"float",

"double"

};

wordIterator = 0;

while (wordIterator < search.length && !checkedFlag) {// пробегаемся по массиву

int poz = 0;// проверяем эквивалентность цепочек

while (search[wordIterator].length() > poz && inCopy.length() > poz) {

if (search[wordIterator].charAt(poz) != inCopy.charAt(poz))

break;

poz++;

}

if (poz == search[wordIterator].length()) {// если цепочка пройдена целиком

checkedFlag = true; // символ найден

inCopy.delete(0, search[wordIterator].length()); // удаляем найденное из строки

tokenString.append('T');// добавляем токен

}

wordIterator++;// К следующему типу

}

}

if (!checkedFlag) {// если за текущую итерацию ничего не удалялось

char currentChar = inCopy.charAt(0);// получаем текущий символ

if (currentChar == ' ') {

tokenString.append(' ');// добавляем пробел в токены

//checkedFlag = true; // символ найден

// удаляем этот и следующие пробелы если они есть

while (inCopy.length() != 0) {

if (inCopy.charAt(0) != ' ') break;

inCopy.deleteCharAt(0);

}

} else if (// A..Z || a..z ||\_||- символ строки

65 <= currentChar && currentChar <= 90 || 97 <= currentChar && currentChar <= 122

|| currentChar == '\_' || currentChar == '-') {

if (tokenString.charAt(tokenString.length() - 1) == 'T') {

// если до этого был тип без пробела, затирем

tokenString.deleteCharAt(tokenString.length() - 1);

tokenString.append('I');

} else if (tokenString.charAt(tokenString.length() - 1) != 'I') {

// если до этого идентификатор сливаем их вместе

tokenString.append('I');

}

//checkedFlag = true; // символ найден

inCopy.deleteCharAt(0);// удаляем текущий символ

} else if (48 <= currentChar && currentChar <= 57) {// 0..9 символ строки

if (tokenString.charAt(tokenString.length() - 1) == 'T') {

// если до этого был тип без пробела, затирем

tokenString.deleteCharAt(tokenString.length() - 1);

tokenString.append('I');

} else if (tokenString.charAt(tokenString.length() - 1) != 'I') {

// если до этого не стоял другой символ, идентификатор

throw new Exception("Identifier cannot start from number! \"" + currentChar + "\"");

}

//checkedFlag = true; // символ найден

inCopy.deleteCharAt(0);// удаляем текущий символ

} else if ('(' == currentChar || ')' == currentChar ||

',' == currentChar || ';' == currentChar) {

tokenString.append(currentChar);

//checkedFlag = true; // символ найден

inCopy.deleteCharAt(0);// удаляем текущий символ

} else {// если найденный символ не известен

throw new Exception("String contains forbidden symbol! \"" + currentChar + "\"");

}

}

}

return tokenString.deleteCharAt(0).toString();

}

Функция, реализующая конечный автомат:

static int[][] stateCodes = {

//P T I V ( , );

/\*1\*/{2, 3, 0, 3, 0, 0, 0},

/\*2\*/{0, 3, 0, 0, 0, 0, 0},

/\*3\*/{0, 0, 4, 0, 0, 0, 0},

/\*4\*/{0, 0, 0, 0, 5, 0, 0},

/\*5\*/{6, 7, 0, 0, 0, 0, 10},

/\*6\*/{0, 7, 0, 0, 0, 0, 0},

/\*7\*/{0, 0, 8, 0, 0, 0, 0},

/\*8\*/{0, 0, 0, 0, 0, 9, 10},

/\*9\*/{6, 7, 0, 0, 0, 0, 0}

};

static boolean isPrototypeInTokensLine(String tokens) throws Exception {

int qState = 1;// состояние автомата

StringBuilder tokensCopy = new StringBuilder(tokens);

while (qState != 10 && qState != 0 && tokensCopy.length() != 0) {

switch (tokensCopy.charAt(0)) {

case 'P':

if (stateCodes[qState - 1][0] == 0) {// ошибка

throw new Exception("Parse error! Code:[symbol->'P', state->" + qState + "]");

} else {

qState = stateCodes[qState - 1][0];// меняем состояние на следующее

}

break;

case 'T':

if (stateCodes[qState - 1][1] == 0) {// ошибка

throw new Exception("Parse error! Code:[symbol->'T', state->" + qState + "]");

} else {

qState = stateCodes[qState - 1][1];// меняем состояние на следующее

}

break;

case 'I':

if (stateCodes[qState - 1][2] == 0) {// ошибка

throw new Exception("Parse error! Code:[symbol->'I', state->" + qState + "]");

} else {

qState = stateCodes[qState - 1][2];// меняем состояние на следующее

}

break;

case 'V':

if (stateCodes[qState - 1][3] == 0) {// ошибка

throw new Exception("Parse error! Code:[symbol->'V', state->" + qState + "]");

} else {

qState = stateCodes[qState - 1][3];// меняем состояние на следующее

}

break;

case '(':

if (stateCodes[qState - 1][4] == 0) {// ошибка

throw new Exception("Parse error! Code:[symbol->'(', state->" + qState + "]");

} else {

qState = stateCodes[qState - 1][4];// меняем состояние на следующее

}

break;

case ',':

if (stateCodes[qState - 1][5] == 0) {// ошибка

throw new Exception("Parse error! Code:[symbol->',', state->" + qState + "]");

} else {

qState = stateCodes[qState - 1][5];// меняем состояние на следующее

}

break;

case ')':

if (tokensCopy.length() > 1) {

if (tokensCopy.charAt(1) == ';') {

if (stateCodes[qState - 1][6] == 0) {// ошибка

throw new Exception("Parse error! Code:[symbol->')', state->"

+ qState + "]");

} else {

qState = stateCodes[qState - 1][6];// меняем состояние на следующее

}

} else {

throw new Exception("Parse error! Code:[symbol->')', state->"

+ qState + "] - \";\" error2");

}

} else {

throw new Exception("Parse error! Code:[symbol->')', state->"

+ qState + "] - \";\" error1");

}

break;

default:

// тк строка уже разбита на токены, пробелы можно игнорировать

}

// удаляем обработанный символ

tokensCopy.deleteCharAt(0);

}

if (tokensCopy.length() == 0 && qState != 10) {// не найден символ завершения

throw new Exception("Parse error! no \");\" ");

}

return true;

}

Класс запуска программы:

import java.util.Scanner;

public class Main {

static Scanner scan = new Scanner(System.in);

static String inLine;// вводимая строка

public static void main(String[] args) {

System.out.println("Enter string:");

inLine = scan.nextLine().trim();// ввод строки

while (!inLine.equals("end")) {

try {// ставим блок отлова ошибок(проставленных мною) разбора строки

String token = getTokenString(inLine);// получаем строку токенов из исходной строки

System.out.println("\nToken string: " + token);

if (isPrototypeInTokensLine(token)) {// анализируем

System.out.println("\nprototype correct");

}

} catch (Exception e) {

System.out.println("Error: " + e.getMessage());

//e.printStackTrace();

}

System.out.println("\n\nEnter string:");

inLine = scan.nextLine().trim();// ввод строки

}

}

}

**Работа программы с тестовыми данными приведена в таблице 2 и проиллюстрирована на рисунке 2:**

|  |  |  |
| --- | --- | --- |
| Исходные данные | Ожидаемый результат | Полученный результат |
| **int as3(int a\_34, float bjt, unsigned char car);** | T I(T I, T I, P T I);  true | Token string:  T I(T I, T I, P T I);  prototype correct |
| **void b();** | V I();  true | Token string:  V I();  prototype correct |
| **int b(int a\_34,** | Error | Token string:  T I(T I,  Error: Parse error! no ");" |
| **Void& b();** | Error | Error: String contains forbidden symbol! "&" |

Таблица 2 – Работа программы

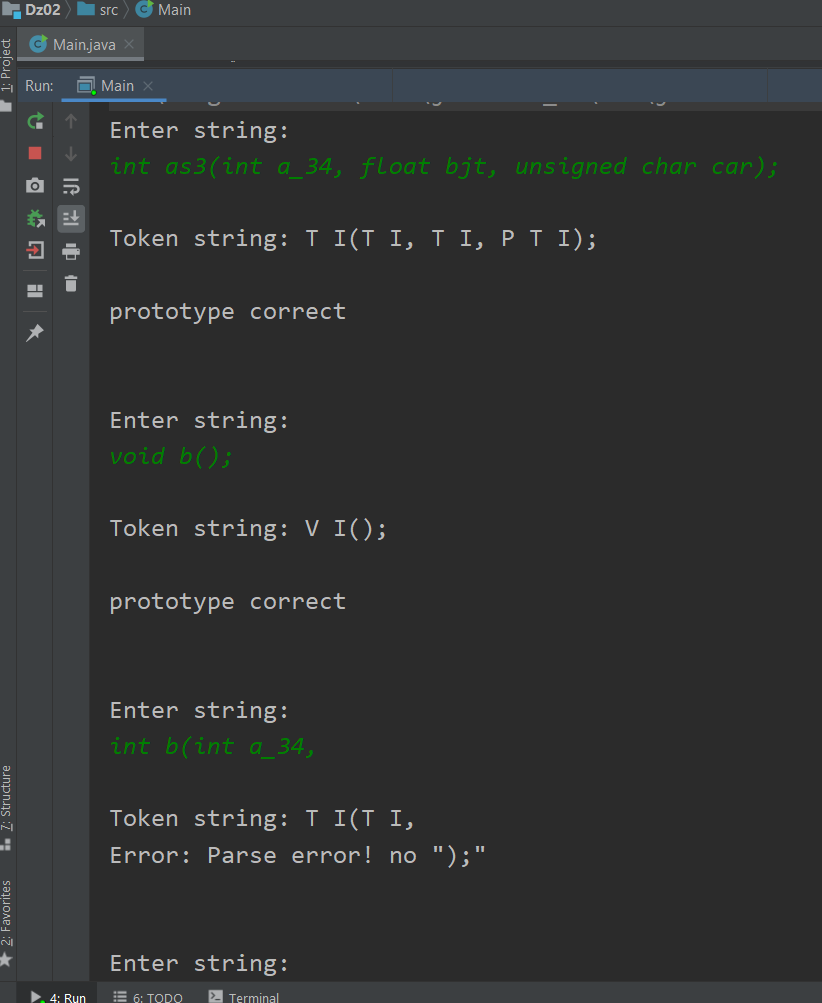


Рисунок 2 – Вывод программы при различных входных данных

**Контрольные вопросы:**

1) Дайте определение формального языка и формальной грамматики.

Формальным языком L в алфавите A называют произвольное подмножество множества A\*. Таким образом, язык определяется как множество допустимых предложений.

Формальная грамматика – это математическая система, определяющая язык посредством порождающих правил – правил продукции. Она определяется как четверка: G = (VT, VN, P, S),

2) Как определяется тип грамматики по Хомскому?

– Тип 0, грамматики фразовой структуры или грамматики «без ограничений»: в таких грамматиках допустимо наличие любых правил вывода, что свойственно грамматикам естественных языков;

– Тип 1, контекстно-зависимые (неукорачивающие) грамматики:в этих грамматиках для правил вида α X β → α x β возможность подстановки строки х вместо символа X определяется присутствием контекста, что также свойственно грамматикам естественных языков;

– Тип 2, контекстно-свободные грамматики: подстановки не зависят от контекста;

– Тип 3, регулярная грамматика: расширение допускает не более одного правила вида S → e, но в этом случае аксиома не должна появляться в правых частях правил

3) Поясните физический смысл и обозначения формы Бэкуса – Наура.

Форма Бэкуса-Наура (БНФ). БНФ связывает терминальные и нетерминальные символы, используя две операции: «::=» – «можно заменить на»; «|» – «или». Основное достоинство – группировка правил, определяющих каждый нетерминал. Нетерминалы при этом записываются в угловых скобках

4) Что такое лексический анализ? Какие методы выполнения лексического анализа вы знаете?

Лексический анализ – процесс преобразования исходного текста в строку однородных символов. Каждый символ результирующей строки – токен соответствует слову языка – лексеме и характеризуется набором атрибутов, таких как тип, адрес и т. п., поэтому строку токенов часто представляют таблицей, строка которой соответствует одному токену.

Методы:

а) При помощи конечных автоматов

б) Восходящий анализ

в) Низходящий анализ

5) Что такое синтаксический анализ? Какие методы синтаксического анализа вы знаете? К каким грамматикам применяются перечисленные вами методы?

Синтаксический анализ – процесс распознавания конструкций языка в строке токенов. Главным результатом является информация об ошибках в выражениях, операторах и описаниях программы. Построение конечного автомата, получающего на вход строку лексем

Метод рекурсивного спуска. Метод рекурсивного спуска основывается на синтаксических диаграммах языка. Согласно этому методу для каждого нетерминала разрабатывают рекурсивную процедуру. Основная программа вызывает процедуру аксиомы, которая вызывает процедуры нетерминалов, упомянутые в правой части аксиомы и т. д. В эти же процедуры встраивают семантическую обработку распознанных конструкций.

6) Что является результатом лексического анализа?

При лексическом разборе будет получена таблица токенов и, возможно, расширены таблицы переменных и литералов

7) Что является результатом синтаксического анализа?

Результатом является информация об ошибках в выражениях, операторах и описаниях программы.

8) В чем заключается метод рекурсивного спуска?

Метод рекурсивного спуска основывается на синтаксических диаграммах языка. Согласно этому методу для каждого нетерминала разрабатывают рекурсивную процедуру. Основная программа вызывает процедуру аксиомы, которая вызывает процедуры нетерминалов, упомянутые в правой части аксиомы и т. д. В эти же процедуры встраивают семантическую обработку распознанных конструкций.

9) Что такое таблица предшествования и для чего она строится?

В таблицу сводятся отношения предшествования терминалов (знаков операций), полученных с учетом приоритетов операций. После чего возможно будет четко определить начало или конец основы

10) Как с использованием таблицы предшествования осуществляют синтаксический анализ?

Из входной строки выделяются символы операций, после чего в соответствии с таблицей для пар символов определяются отношения, чтобы затем свертывать выражение в соответствии с отношениями и заменить на результат. Эта операция повторяется в цикле пока не будет найдена ошибка или строка не будет преобразована.

**Вывод:**

Я закрепил знания теоретических основ и основных методов и приемов разработки лексических и синтаксических анализаторов регулярных и контекстно-свободных формальных языков. И применил полученные навыки для написания решения поставленной задачи.