МИНИСТЕРСТВО НАУКИ И ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

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

Кафедра вычислительной техники



**ОТЧЁТ**

**по лабораторной работе №4**

*Синтаксис языков программирования. Нисходящий синтаксический анализ*

**по дисциплине:** *Теория формальных языков и компиляторов*

**Вариант: 42411412**

Выполнила:Проверил:

Студентка гр. АВТ-709, АВТФ *Доцент*

*Андерсон Д. В. Малявко А. А.*

«\_\_\_» \_\_\_\_\_\_ 2020 г.«\_\_\_» \_\_\_\_\_\_ 2020 г.

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

(подпись) (подпись)

Новосибирск

2020

**Оглавление**

[**Введение** 3](#_Toc38035668)

[**Вариант** 4](#_Toc38035669)

[**Ход работы** 5](#_Toc38035670)

[*1.* *Разработка грамматики* 5](#_Toc38035671)

[*2.* *Приведение и проверка грамматики на принадлежность к классу LL(1)* 6](#_Toc38035672)

[*3.* *Описание грамматики разрабатываемого языка* 8](#_Toc38035673)

[*4.* *Построение нисходящего синтаксического акцептора* 10](#_Toc38035674)

[**Вывод** 13](#_Toc38035675)

## **Введение**

*Цели работы*: изучение основных идей и понятий нисходящих методов синтаксического анализа, выявление свойств формальных грамматик, необходимых для реализации нисходящего восстановления дерева грамматического разбора, приобретение навыков построения процедурной и различных автоматных реализаций нисходящего анализа, исследование поведения нисходящих синтаксических акцепторов.

*Порядок выполнения работы:*

1. Используя пакет ВебТрансЛаб:

* расширить грамматику заданного на курсовую работу языка до полной;
* изучить и освоить проверку принадлежности грамматики к классу LL1, используя в качестве проверяемых грамматики, полученные при выполнении работы №4;
* освоить технологию удаления символов из множеств выбора правил с использованием тега <exclude> для приведения грамматики к классу LL1 и примеров 6IfElseWithConflict и 7IfElseNoConflict;
* построить конечный автомат со стековой памятью и несколькими состояниями (шаблон …SyntAsMultiFSM…), разобраться в структуре управляющей таблицы автомата, уяснить способы формирования и использования всех полей;
* построить конечный автомат со стековой памятью и одним состоянием, управляемый входным символом и символом, снятым с верхушки стека (шаблон …SyntAsSingleFSM…), разобраться в структуре управляющей таблицы автомата, уяснить способы формирования и использования клеток таблицы;
* построить процедурную реализацию рекурсивного спуска (шаблон …SyntAsRD…), уяснить способы формирования функций этого акцептора.

1. Выполнить трассировку процессов нисходящего синтаксического акцепта, изучить поведение всех построенных синтаксических акцепторов при разборе как правильных предложений, так и предложений с намеренно внесенными синтаксическими ошибками.
2. Проанализировать и сравнить между собой все полученные тексты программ и результаты выполнения пункта 3.2. Оценить степень пригодности изученных вариантов реализации нисходящих синтаксических акцепторов для выполнения курсовой работы.
3. Подготовить, сдать и защитить отчет к лабораторной работе.

## **Вариант**

|  |  |
| --- | --- |
| Идентификаторы | $<пЦ><пБ>  ($1ad, $74c, $0B…) |
| Константы | целые по основаниям 2,8 и 10;  вещественные;  символьные |
| Объявление примитивных типов | Целое – card[inal][\_u]  Вещественное – double  Символьное – litera |
| Оператор присваивания | <И> **=** <В>; |
| Условный оператор | **at** <ЛВ> **do** <ОБ> [**else do** <ОБ>] |
| Оператор цикла | **exec** <ОБ> **with** <И> **from** <К> **to** <К>[**step** <К>] |
| Оператор переключателя | **select** <B> **case (**<К>**) <**ОБ**> [break;]**…**[case()** <ОБ>**] end** |
| Формат псевдокода | Пентады <М><Код><Оп><Оп><Р> |

Обозначения:

**[...]** – необязательная часть конструкции;

**…** – предшествующая часть конструкции повторяется произвольное количество раз;

< > – описание сокращений:

<пБ>|<пЦ> – непустая последовательность букв | непустая последовательность цифр

<И> – Идентификатор (имя переменной / объекта); <К> – Константа;

<В> – произвольное Выражение;

<ЛВ> – Логическое Выражение;

<ОБ> – Оператор или Блок;

<О> – одиночный оператор;

<ОП> – оператор присваивания;

<Код> – поле кода операции;

<М> – поле метки;

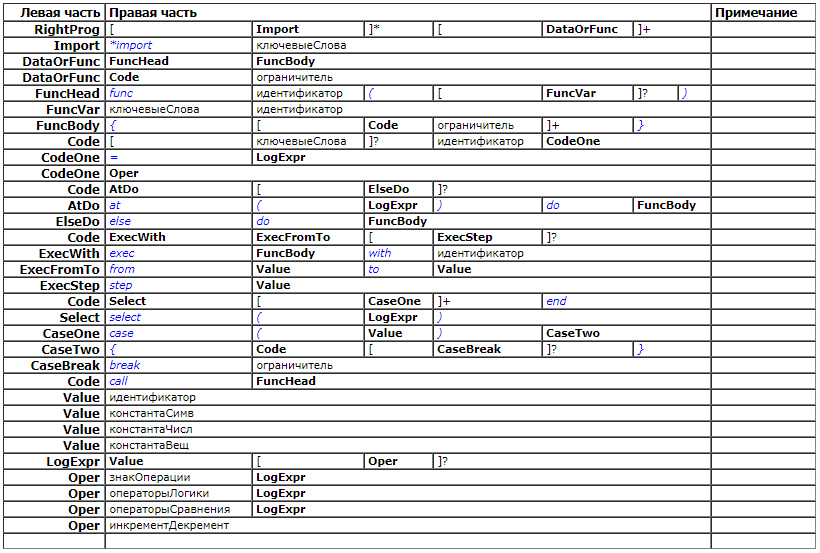
<Оп> – поле наименования операнда;

<Р> – поле наименования результата

## **Ход работы**

## *Разработка грамматики*

В данной лабораторной работе были исправлены правила синтаксиса из лабораторной работы №3. Грамматика была доработана и расширена, в соответствии с технологией «снизу-вверх».

Рисунок 1. Разработанная грамматика

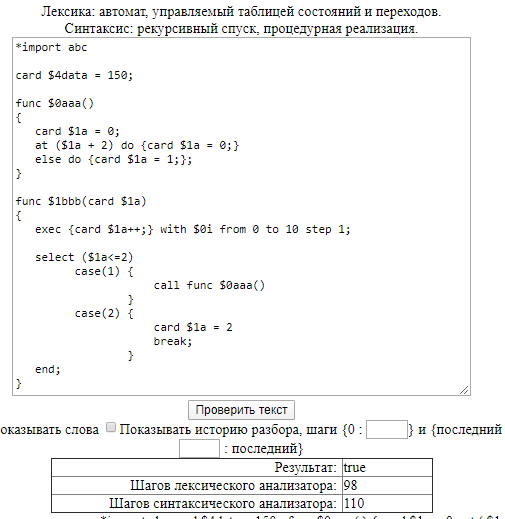


Рисунок 2. Проверка правильности грамматики

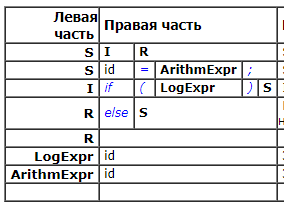
## *Приведение и проверка грамматики на принадлежность к классу LL(1)*

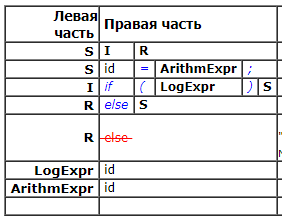
Из примеров 6 (рисунок 3.1) и 7 (рисунок 3.2) видно различие в непринадлежности грамматики к классу LL(1) и ее принадлежности соответственно.

Рисунок 3.1. Пример 6

Рисунок 3.2. Пример 7

По сути, грамматика из примера 6 – та же грамматика, что и в примере 7, но с одним изменением. В примере 7 с помощью тега <exclude> было удалено повторение (конфликт), которое не позволяла грамматике из примера 6 быть принадлежной к классу LL(1) (рисунок 4.1, 4.2).

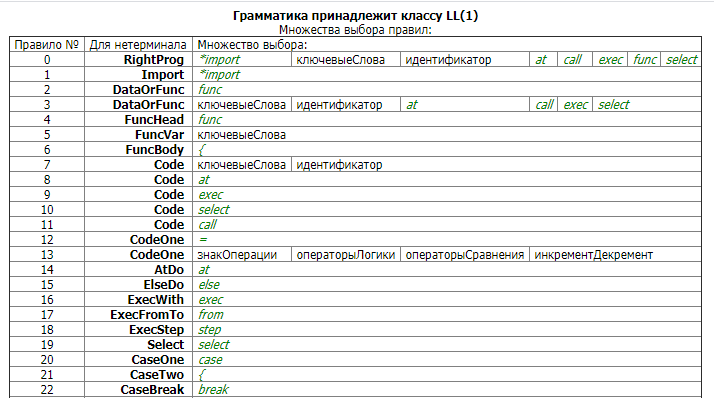
Рисунок 4.1. Пример 6

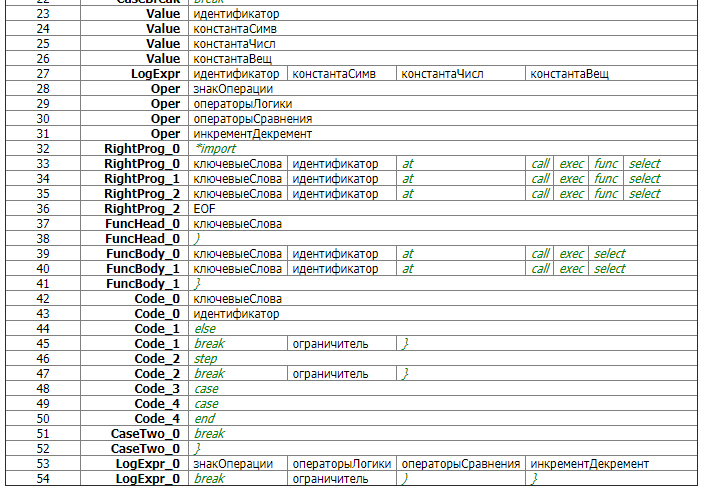
Рисунок 4.1. Пример 7

LL(1)-грамматикой называется такая контекстно-свободная грамматика, у которой множества выбора правил одинаковыми нетерминалами в левой части попарно не пересекаются. Принято считать, что символы в названии класса LL(1)-грамматик обозначают:

* Первая буква L (left) – чтение предложения слева-направо;
* Вторая буква L(leftmost) – на каждом шаге принимается решение о замене самого левого нетерминала из текущего уровня
* Цифра 1 в скобках указывает на количество символов из начала предложения, необходимых для принятия решения о выборе правила на каждом шаге детерминированного дерева.

На рисунках 5.1 – 5.2 представлено множество выбора правил для грамматики разрабатываемого языка ADV. Как видно из рисунка, грамматика принадлежит классу LL(1).

Рисунок 5.1. Множество правил выбора (1)

Рисунок 5.2. Множество правил выбора (2)

## *Описание грамматики разрабатываемого языка*

Правильная программа на языке ADV может начинаться с подключения сторонних библиотек или файлов. Для этой цели есть специальное ключевое слово ***\**import**, после которого идет имя подключаемой библиотеки/файла.

|  |  |
| --- | --- |
| Объявление | \*import *имя\_файла* |
| Пример | \*import rules |

После блока с подключением файлов, в программе должно идти объявление переменных или функций. Объявление переменной:

|  |  |
| --- | --- |
| Объявление | *тип\_переменной* *имя\_переменной* |
| Пример | cardinal $4number, litera $0lit, double $5drob |

Тип переменной – один из стандартных типов заданного языка (**card(inal(\_u))**, **double**, **litera**).

Функция в данном разрабатываемом языке объявляется следующим образом:

|  |  |
| --- | --- |
| Объявление | func *имя\_функции* (…, *параметр*) {  *тело\_функции*  } |
| Пример | func $0main (litera $1lit){  card $1num  } |

Объявление начинается с помощью ключевого слова **func**, далее следует имя функции и принимаемые параметры, заключенные в круглые скобки. Важно заметить, что передаваемых в функцию параметров может и не быть. Тело функции обязательно заключается в фигурные скобки.

Внутри функции могут быть использованы операторы присваивания, цикла, условный оператор и оператор переключения. Рассмотрим их подробнее.

Оператор присваивания объявляется следующим образом:

|  |  |
| --- | --- |
| Объявление | *тип\_переменной* *имя\_переменной* = [*имя\_переменной*, *выражение*, вещественная\_константа, числовая\_константа, символьная\_константа] |
| Пример | card $1num = 709  litera $1lit = ‘I love avtf’ |

Оператор цикла объявляется начинается с ключевого слова **exec**, после которого следует оператор или блок действий, далее, после ключевого слова **with** следует идентификатор счетчика, с помощью ключевых слов **from** и **to** указываются границы изменения переменной счетчика, далее, опционально, после ключевого слова **step** идет шаг, с которым будет меняться значение переменной счетчика.

|  |  |
| --- | --- |
| Объявление | exec { *последовательность*\_*действий* } with *имя\_переменной* from *константа* to *константа* step *константа* |
| Пример | exec { $1a ++ } with $0i from 0 to 10 step 1 |

Условный оператор в языке ADV объявляется так: с помощью ключевого слова **at** задается некоторое логическое выражение, после чего идет ключевое слово **do**, предшествующее первому оператору или блоку действий. Далее, опционально, после пары ключевых слов **else do**, следует другой блок действий или оператор.

|  |  |
| --- | --- |
| Объявление | at (*логическое\_выражение*) do { *последовательность\_действий\_1* } else do { *последовательность\_действий\_2* } |
| Пример | at ($1a + 2) do {card $1a = 0}  else do {card $1a = 1} |

Оператор переключения: в начале идет ключевое слово **select** после которого в скобках следует произвольное выражение. Далее может идти любое количество блоков, выглядящих следующим образом: ключевое слово **case**, в скобках значение, которое принимает выражение из предыдущего шага, в фигурных скобках оператор или блок действий (опционально ключевое слово **break;**). После всех описанных блоков, оператор переключения обязательно должен закрываться ключевым словом **end**.

|  |  |
| --- | --- |
| Объявление | select (*выражение*)  case(*константа*) { *последовательность\_действий\_1* }  case(*константа*) { *последовательность\_действий\_2* }  end |
| Пример | select ($1a<2)  case(1) { card $1a = 0 }  case(2) { card $1b = 2  break; }  end |

Вызов функции в данном языке происходит следующим образом: на вызов функции указывает ключевое слово **call**, после которого пишется ключевое слово **func**, имя функции и принимаемые параметры (если есть).

|  |  |
| --- | --- |
| Объявление | call func *имя\_функции* (…, *параметр*) |
| Пример | call func $0count()  call func $0maker(double $1drob) |

## *Построение нисходящего синтаксического акцептора*

LL(1)-грамматика может быть преобразована в программу синтаксического акцептора, реализующую рекурсивный спуск. Такая реализация называется процедурной в отличии от автоматных (таблицы). Рекурсивный спуск может быть реализован только на таком языке программирования, который допускает рекурсивный вызов функции.

Функционирование конечного автомата со стековой памятью и несколькими состояниями определяется управляющей таблицей. С каждым состоянием должны быть также связаны операции управление стековой памятью (занесение адреса возврата, снятие адреса с верхушки стека и переключение в состояние возврата) и операция управления чтения символа. Все операции управления могут задаваться булевскими значениями true/false – флажками:

* флажок *a* управляет чтением следующего входного символа;
* флажок *s* управляет занесением адреса точки возврата в стек;
* флажок *r* обеспечивает переключение автомата в состояние, номер которого снимается с верхушки стека возвратов;
* флажок *e* запрещает останов по ошибке, когда состояние соответствует нетерминалу из левой части и есть еще хотя бы одно правило для такого нетерминала.

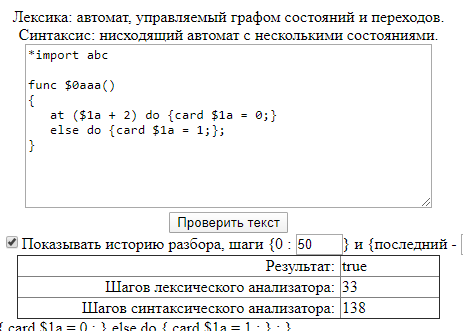


Рисунок 6. Правильная программа

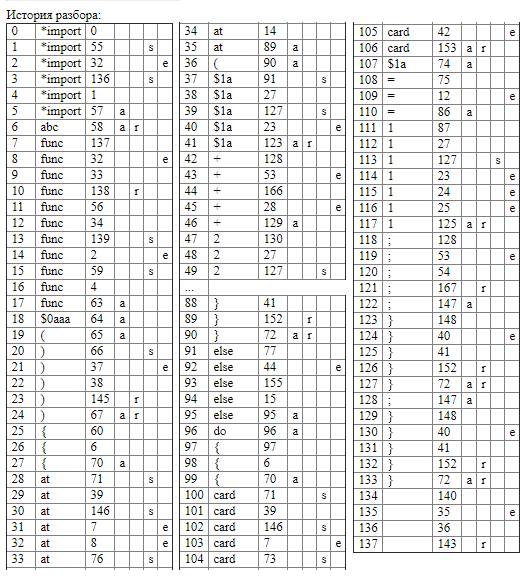


Рисунок 7. История разбора программы

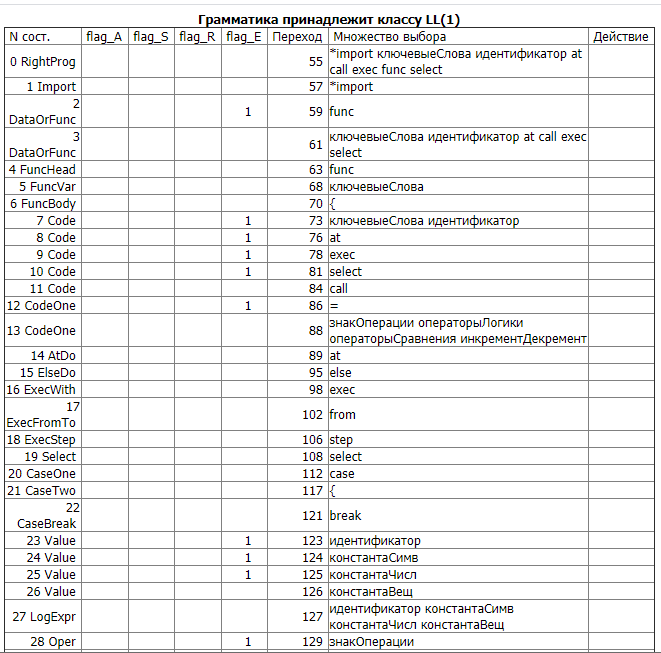


Рисунок 8. Фрагмент управляющей таблицы нисходящего автомата с несколькими состояниями

Грамматику LL(1) можно также преобразовать в конечный автомат с единственным состоянием и стековой памятью, управляемый:

* Текущим входным символом;
* Символом, находящимся на верхушке стека.

Поведение такого автомата определяется управляющей таблицей, столбцы которой соответствуют входным символам, строки – символам, которые могут находится в стеке, а в клетках указывается некоторая последовательность операций над стеком, входным потоком и состоянием автомата.

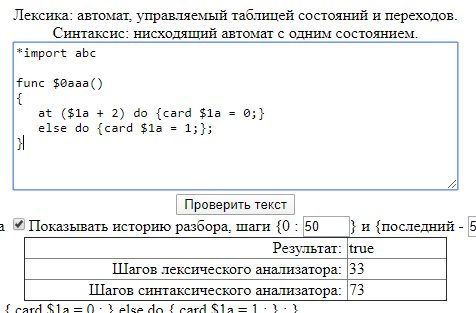


Рисунок 10. Правильная программа

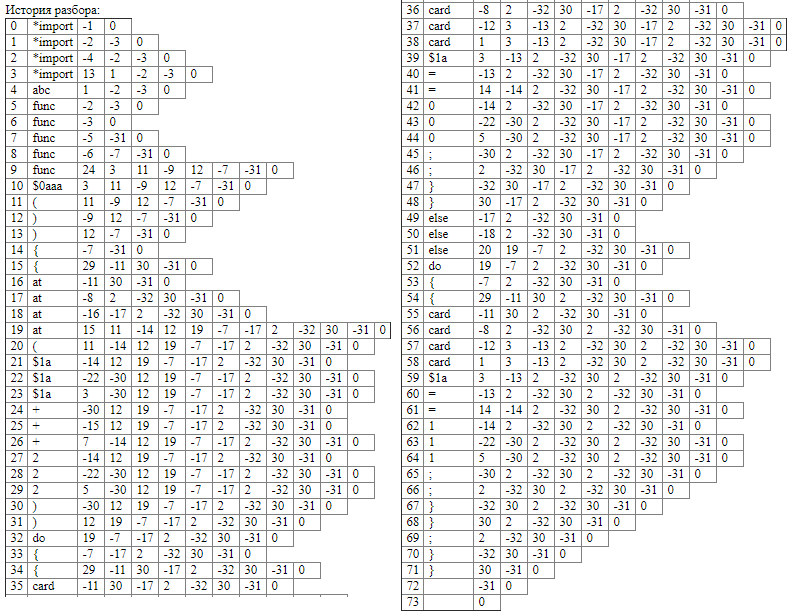


Рисунок 11. История разбора программы

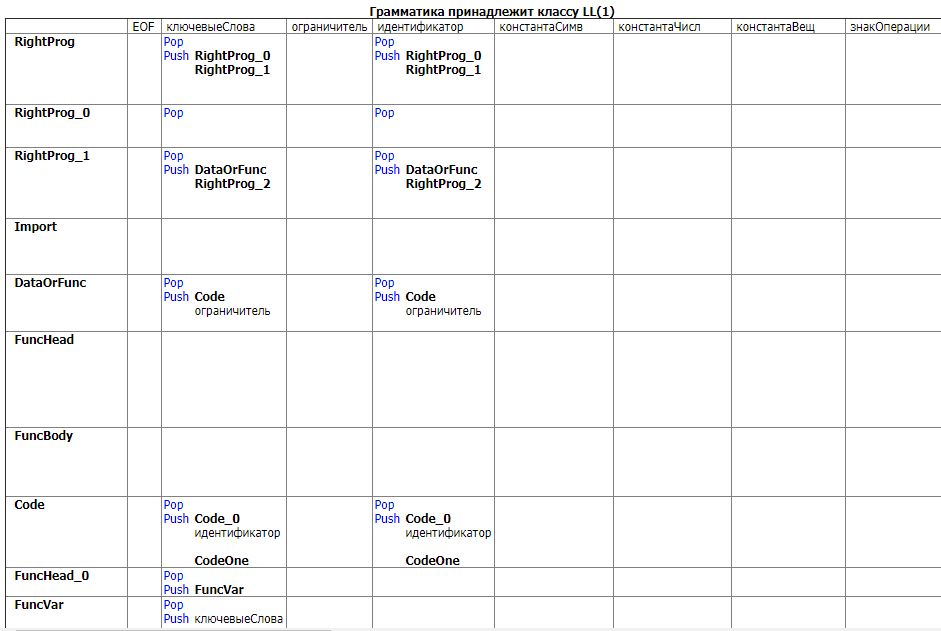


Рисунок 12. Фрагмент управляющей таблицы нисходящего автомата с одним состоянием

Обозначения:

* Pop Снять верхний символ со стека
* Push Поместить в стек цепочку символов, находящуюся в клетке справа от слова Push (верхний символ цепочки становится верхним символом стека)
* Next Прочитать следующий терминал со входа
* Stop Останов по окончанию восстановления дерева разбора правильного предложения
* Пустая клетка Останов по ошибке

## **Вывод**

В результате выполнения лабораторной работы была доработана часть грамматики программируемого языка в соответствии с вариантом. Были изучены основные идеи и понятия нисходящих методов синтаксического анализа, свойства формальных грамматик. Были приобретены навыки построения процедурной и различных автоматных реализаций нисходящего анализа.

Анализируя построенные автоматы и историю разбора написанной тестовой программы (рисунок 7, 11), можно заметить, что автомат с одним состоянием справился обработал программу за меньшее количество шагов, чем автомат с множеством состояний. Из этого следует, что в данном конкретном примере автомат с одним состоянием оказался более эффективен.