Кафедра вычислительной техники



**ЛАБОРАТОРНАЯ РАБОТА №4**

**по дисциплин**е: теория формальных языков и компиляторов  
**на тему:** «Синтаксис языков программирования. Нисходящий синтаксический анализ»

**вариант:** 44242414

|  |  |
| --- | --- |
| Выполнил: | Проверил: |
| Студент гр. *АВТ-709*, *АВТФ* | *Доцент каф. ВТ* |
| *Кустов И.С.* | *Малявко А.А.* |
| «\_\_\_» \_\_\_\_\_\_ 20\_\_г. | «\_\_\_» \_\_\_\_\_\_ 20\_\_г. |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| (подпись) | (подпись) |

**Оглавление**

[1. Цель работы: 3](#_Toc40874216)

[2. Задание: 3](#_Toc40874217)

[3. История работы лексического анализатора: 4](#_Toc40874217)

[4. Ход работы: 6](#_Toc40874218)

[Ввод: 15](#_Toc40874219)

1. Цель 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.

1. Задание:

* 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.

* 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.
* 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.
* Подготовить, сдать и защитить отчет к лабораторной работе.
* Требования к содержанию отчета.

Отчет должен содержать:

-     цель работы;

-     реализацию LL(1)-грамматики для языка, заданного на курсовую работу;

-     описание этой грамматики в качестве фрагмента расчетно-пояснительной записки к курсовой работе;

-     текст процедурной реализации и управляющие таблицы автоматных реализаций нисходящего синтаксического акцептора, построенных по этой грамматике, описание алгоритмов работы соответствующих автоматов;

-     фрагменты историй работы процедурной и автоматных реализаций нисходящего синтаксического акцептора для правильного и ошибочного тестовых примеров с объяснением принципов работы каждого акцептора;

-     выводы и заключение.

**3. История работы лексического анализатора**

Рассмотри принцип работы анализатора на примере строчки:

ifnot return $1a;

* Конечный автомат, управляемый графом состояний и переходов

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Номер шага | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Номер состояния | 0 | 3 | 3 | 3 | 3 | 3 | -3 (KeyWord) |
| Входной символ | i | f | n | o | t | “ “ | “ “ |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Номер шага | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| Номер состояния | 0 | 12 | -8 (Space) | 0 | 3 | 3 | 3 | 3 | 3 | 3 | -3  (KeyWord) |
| Входной символ | “ “ | r | r | r | e | t | u | r | n | “ “ | “ “ |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Номер шага | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| Номер состояния | 0 | 12 | -8 (Space) | 0 | 16 | 25 | 29 | -10(Identifier) | 0 |
| Входной символ | “ “ | $ | $ | $ | 1 | a | ; | ; | ; |

|  |  |  |
| --- | --- | --- |
| Номер шага | 28 | 29 |
| Номер состояния | 5 | -5(Delimiter) |
| Входной символ |  |  |

* Конечный автомат, управляемый таблично

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Номер шага | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Номер состояния | 0 | 3 | 3 | 3 | 3 | 3 | -3 (KeyWord) |
| Входной символ | i | f | n | o | t | “ “ | “ “ |

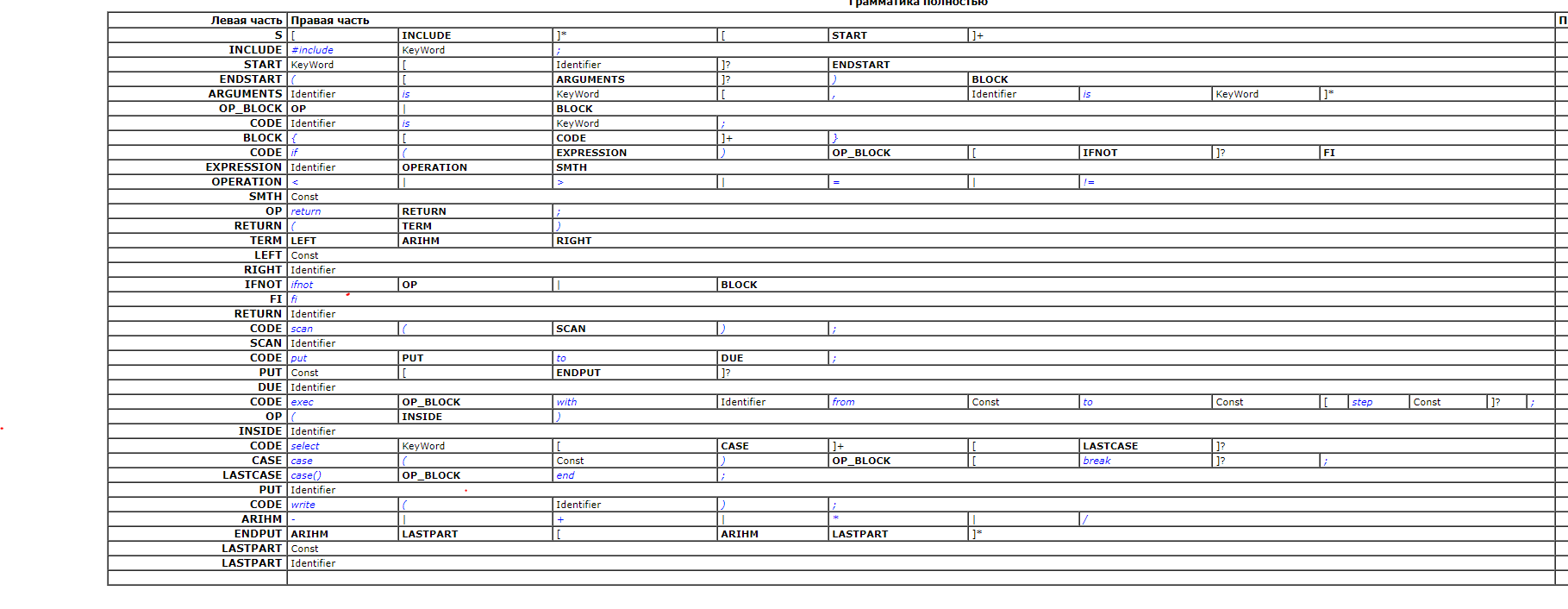
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Номер шага | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| Номер состояния | 0 | -8 (Space) | 0 | 3 | 3 | 3 | 3 | 3 | 3 | -3  (KeyWord) |
| Входной символ | “ “ | r | r | e | t | u | r | n | “ “ | “ “ |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Номер шага | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| Номер состояния | 0 | -8 (Space) | 0 | 12 | 20 | 23 | -10(Identifier) | 0 |
| Входной символ | “ “ | $ | $ | 1 | a | ; | ; | ; |

|  |  |
| --- | --- |
| Номер шага | 26 |
| Номер состояния | -5(Delimiter) |
| Входной символ |  |

1. Ход работы:

LL(1)-грамматика для языка (файл 4.xml)



*Рис.1 – Грамматика языка*

**Описание грамматики:**

LL(1)-грамматикой называется такая контекстно-свободная грамматика, у которой множества выбора правил с одинаковым нетерминалом в левой части попарно не пересекаются.

Любая такая грамматика может быть использована для организации нисходящего детерминированного восстановления дерева грамматического разбора предложений порождаемого ею языка. Другими словами, на основе любой LL(1) -грамматики может быть построен детерминированный нисходящий синтаксический акцептор, проверяющий правильность предложений языка.

Принято считать, что символы в названии класса LL(1)-грамматик обозначают следующее.

Первая буква L (сокращение слова left - левый) — чтение слов анализируемого предложения производится слева направо.

Вторая буква L, (сокращение слова leftmost — самый левый) — на каждом шаге принимается решение для замены самого левого нетерминала из текущего уровня восстанавливаемого дерева.

Цифра 1 в скобках обозначает количество символов из начала остатка. предложения, необходимых для принятия решения о выборе правила на каждом шаге детерминированного нисходящего восстановления дерева грамматического разбора.

Любая S-грамматика одновременно является LL(1)-грамматикой. Некоторые (но не все) q-грамматики являются LL(1)-грамматиками. Существуют грамматики, не являющиеся ни s-грамматиками, ни q-грамматиками, но относящиеся к классу LL(1). Это грамматики, в которых есть порождающие правила правая часть которых начинается с нетерминального символа, но такие, что множества выбора правил с одинаковым нетерминалом в левой части попарно не пересекаются.



*Рис.2. – Принадлежность грамматики к классу LL(1)*

Проверим синтаксические правила, написанные выше.

#include programm;

int $1g ($1a is int ) {

if ($1a < 0) return (0-$1a);

ifnot return $1a;

fi}

void $1main () {

$1array is int;

$1size is int;

$1i is int;

$1j is int;

$1fisrt is int;

$1second is int;

$1s is int;

scan($1size);

put 100000 to $1s;

exec ($1s) with $1s from 1 to 1000;

select a

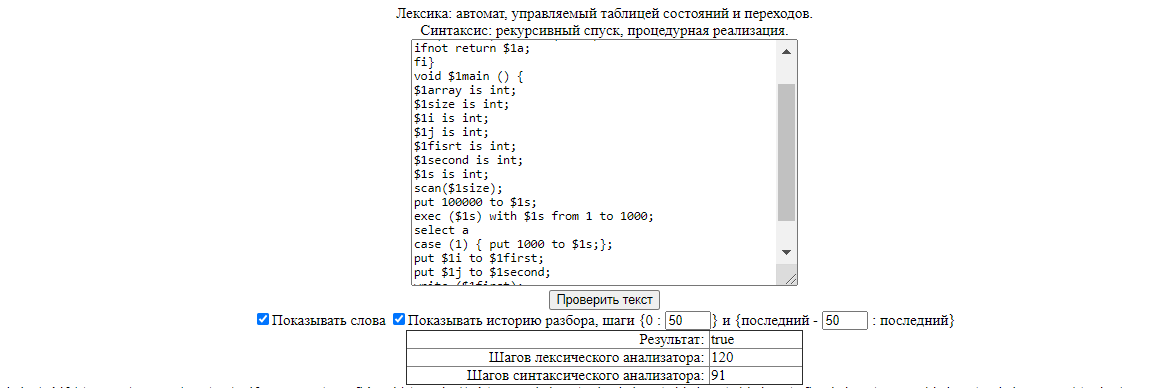
case (1) { put 1000 to $1s;};

put $1i to $1first;

put $1j to $1second;

write ($1first);

write ($1second);

}

*Рис.3. – Результат работы лексического и синтаксического анализаторов*

**Процедурная реализации и управляющие таблицы автоматных реализаций нисходящего синтаксического акцептора:**

Построим таблицу автоматных реализаций.

1. выбираем шаблон lexAsTableSyntAsSingleFSM;
2. при успешном построении лексического и синтаксического анализаторов, показываем упр. таблицу нисх. автомата с одним сост. (файл Excel))
3. выбираем шаблон lexAsTableSyntAsMultiFSM;
4. при успешном построении лексического и синтаксического анализаторов, показываем упр. таблицу нисх. автомата с неск. сост. (файл Excel))

Нисходящими называются такие методы синтаксического акцепта, при которых восстановление дерева протекает сверху (от начального нетерминала S) вниз (к цепочке ɷ).

Каждый шаг процесса восстановления дерева состоит в применении одного непосредственного вывода, т.е. в замене единственного нетерминального символа правой частью какого-либо правила для этого нетерминала. Главное прагматическое требование к этому процессу — необходимость организации безоткатного однонаправленного движения вниз по дереву.

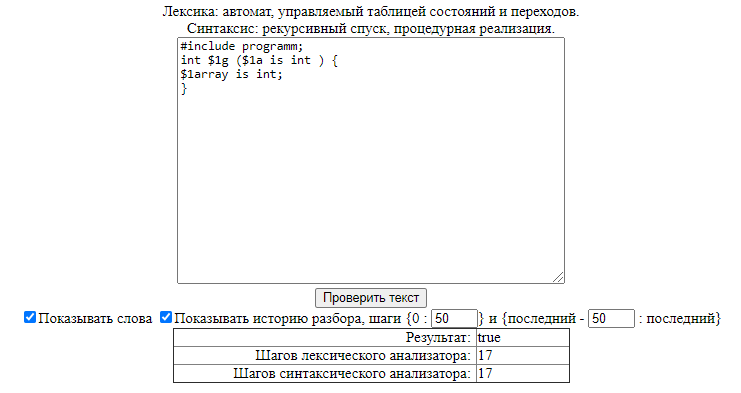
Отсюда следует, что выбор правила на каждом шаге должен осуществляться таким образом, чтобы гарантировать восстановление дерева для любого правильного предложения и обнаружение невозможности это сделать для любого неправильного предложения. Оказывается, что дать такие гарантии можно отнюдь не для любой грамматики и, более того, не для любого языка.

Существуют языки, для которых любая порождающая грамматика не позволяет организовать безвозвратное нисходящее восстановление дерева грамматического разбора (в чистом виде, т. е. без применения специальных мер). Существуют и такие языки, для которых одни порождающие грамматики пригодны для детерминированного нисходящего восстановления дерева, а другие нет.

**Истории работы процедурной и автоматных реализаций нисходящего синтаксического акцептора:**

1. выбираем шаблон lexAsTableSyntAsRD;
2. запускаем построенный транслятор.

Напишем правильный с точки зрения грамматики фрагмент программы.

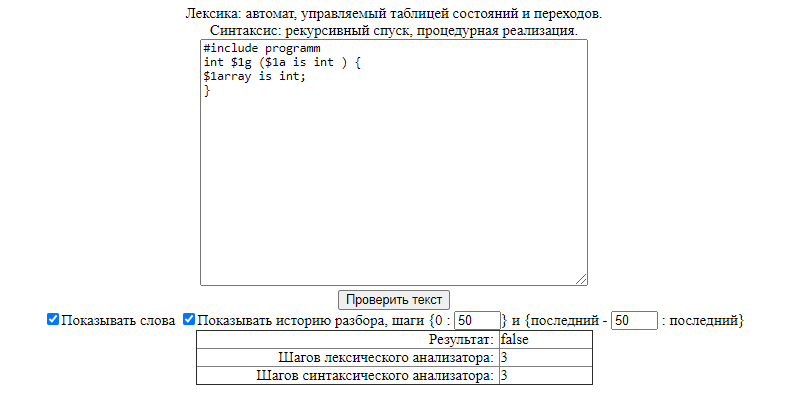


*Рис.4. – Фрагмент программы, правильный с точки зрения грамматики*

*Таблица 1. – История разбора фрагмента*

|  |  |  |
| --- | --- | --- |
| 0 | #include | S |
| 1 | #include | S\_0 |
| 2 | #include | INCLUDE |
| 3 | int | S\_0 |
| 4 | int | S\_1 |
| 5 | int | START |
| 6 | int | START\_0 |
| 7 | abs | START\_1 |
| 8 | ( | START\_1 |
| 9 | ( | ENDSTART |
| 10 | $1a | ENDSTART\_0 |
| 11 | $1a | ARGUMENTS |
| 12 | ) | ARGUMENTS\_0 |
| 13 | ) | ENDSTART\_0 |
| 14 | { | OP\_BLOCK |
| 15 | if | BLOCK\_0 |
| 16 | if | CODE |
| 17 | $1a | EXPRESSION |
| 18 | < | OPERATION |
| 19 | 0 | SMTH |
| 20 | return | OP\_BLOCK |
| 21 | ( | RETURN |
| 22 | 0 | TERM |
| 23 | 0 | LEFT |
| 24 | $1a | RIGHT |
| 25 | ifnot | CODE\_0 |
| 26 | ifnot | IFNOT |
| 27 | return | OP |
| 28 | $1a | RETURN |
| 29 | fi | FI |
| 30 | } | BLOCK\_1 |
| 31 |  | S\_2 |

Напишем неправильный с точки зрения грамматики фрагмент программы.



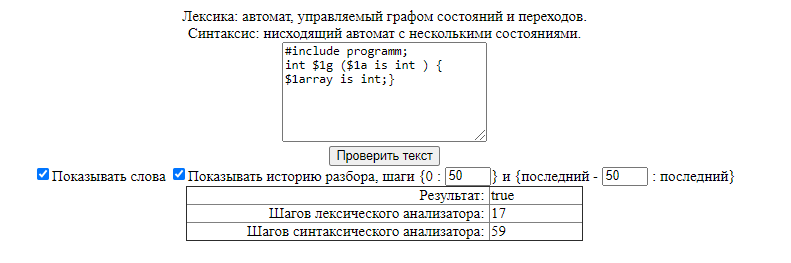
*Рис.5. – Фрагмент программы, неправильный с точки зрения грамматики*

*Таблица 2. – История разбора фрагмента*

|  |  |  |
| --- | --- | --- |
| 0 | #include | S |
| 1 | #include | S\_0 |
| 2 | #include | INCLUDE |

1. выбираем шаблон lexAsTableSyntAsMultiFSM;
2. запускаем построенный транслятор.

Напишем правильный с точки зрения грамматики фрагмент программы.

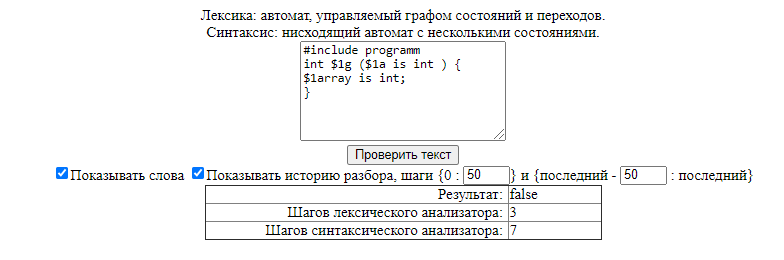


*Рис.6. – Фрагмент программы, правильный с точки зрения грамматики*

*Таблица 3. – История разбора фрагмента*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | #include | 0 |  |  |  |  |
| 1 | #include | 63 |  |  | s |  |
| 2 | #include | 37 |  |  |  | e |
| 3 | #include | 171 |  |  | s |  |
| 4 | #include | 1 |  |  |  |  |
| 5 | #include | 65 | a |  |  |  |
| 6 | programm | 66 | a |  |  |  |
| 7 | ; | 67 | a | r |  |  |
| 8 | int | 172 |  |  |  |  |
| 9 | int | 37 |  |  |  | e |
| 10 | int | 38 |  |  |  |  |
| 11 | int | 173 |  | r |  |  |
| 12 | int | 64 |  |  |  |  |
| 13 | int | 39 |  |  |  |  |
| 14 | int | 174 |  |  | s |  |
| 15 | int | 2 |  |  |  |  |
| 16 | int | 68 |  |  | s |  |
| 17 | int | 42 |  |  |  |  |
| 18 | int | 179 | a |  |  |  |
| 19 | abs | 180 |  |  |  |  |
| 20 | abs | 43 |  |  |  | e |
| 21 | abs | 181 | a |  |  |  |
| 22 | ( | 182 |  |  |  |  |
| 23 | ( | 43 |  |  |  | e |
| 24 | ( | 44 |  |  |  |  |
| 25 | ( | 183 |  | r |  |  |
| 26 | ( | 69 |  |  |  |  |
| 27 | ( | 3 |  |  |  |  |
| 28 | ( | 70 | a |  |  |  |
| 29 | $1a | 71 |  |  | s |  |
| 30 | $1a | 45 |  |  |  | e |
| 31 | $1a | 184 |  |  | s |  |
| 32 | $1a | 4 |  |  |  |  |
| 33 | $1a | 74 | a |  |  |  |
| 34 | is | 75 | a |  |  |  |
| 35 | int | 76 | a |  |  |  |
| 36 | ) | 77 |  |  |  |  |
| 37 | ) | 47 |  |  |  | e |
| 38 | ) | 48 |  |  |  |  |
| 39 | ) | 192 |  | r |  |  |
| 40 | ) | 185 |  |  |  |  |
| 41 | ) | 45 |  |  |  | e |
| 42 | ) | 46 |  |  |  |  |
| 43 | ) | 186 |  | r |  |  |
| 44 | ) | 72 | a |  |  |  |
| 45 | { | 73 |  |  |  |  |
| 46 | { | 5 |  |  |  | e |
| 47 | { | 6 |  |  |  | e |
| 48 | { | 7 |  |  |  |  |
| 49 | { | 84 | a |  |  |  |
| ... |  |  |  |  |  |  |
| 59 | $1a | 130 | a |  |  |  |
| 60 | < | 131 |  |  | s |  |
| 61 | < | 17 |  |  |  | e |
| 62 | < | 133 | a | r |  |  |
| 63 | 0 | 132 |  |  |  |  |
| 64 | 0 | 19 |  |  |  |  |
| 65 | 0 | 135 | a | r |  |  |
| 66 | ) | 94 | a |  |  |  |
| 67 | return | 95 |  |  | s |  |
| 68 | return | 5 |  |  |  | e |
| 69 | return | 78 | a |  |  |  |
| 70 | ( | 79 |  |  | s |  |
| 71 | ( | 22 |  |  |  | e |
| 72 | ( | 142 | a |  |  |  |
| 73 | 0 | 143 |  |  | s |  |
| 74 | 0 | 24 |  |  |  |  |
| 75 | 0 | 146 |  |  | s |  |
| 76 | 0 | 25 |  |  |  |  |
| 77 | 0 | 149 | a | r |  |  |
| 78 | - | 147 | a |  |  |  |
| 79 | $1a | 148 |  |  |  |  |
| 80 | $1a | 26 |  |  |  |  |
| 81 | $1a | 150 | a | r |  |  |
| 82 | ) | 144 | a | r |  |  |
| 83 | ; | 80 | a | r |  |  |
| 84 | ifnot | 96 |  |  | s |  |
| 85 | ifnot | 52 |  |  |  | e |
| 86 | ifnot | 198 |  |  |  |  |
| 87 | ifnot | 27 |  |  |  | e |
| 88 | ifnot | 151 | a |  |  |  |
| 89 | return | 152 |  |  |  |  |
| 90 | return | 20 |  |  |  | e |
| 91 | return | 136 | a |  |  |  |
| 92 | $1a | 137 |  |  | s |  |
| 93 | $1a | 22 |  |  |  | e |
| 94 | $1a | 23 |  |  |  |  |
| 95 | $1a | 145 | a | r |  |  |
| 96 | ; | 138 | a | r |  |  |
| 97 | fi | 97 |  |  |  |  |
| 98 | fi | 29 |  |  |  |  |
| 99 | fi | 154 | a | r |  |  |
| 100 | } | 194 |  |  |  |  |
| 101 | } | 50 |  |  |  | e |
| 102 | } | 51 |  |  |  |  |
| 103 | } | 197 |  | r |  |  |
| 104 | } | 86 | a | r |  |  |
| 105 |  | 175 |  |  |  |  |
| 106 |  | 40 |  |  |  | e |
| 107 |  | 41 |  |  |  |  |
| 108 |  | 178 |  | r |  |  |

Напишем неправильный с точки зрения грамматики фрагмент программы.



*Рис.7. – Фрагмент программы, неправильный с точки зрения грамматики*

*Таблица 4. – История разбора фрагмента*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | #include | 0 |  |  |  |  |
| 1 | #include | 63 |  |  | s |  |
| 2 | #include | 37 |  |  |  | e |
| 3 | #include | 171 |  |  | s |  |
| 4 | #include | 1 |  |  |  |  |
| 5 | #include | 65 | a |  |  |  |
| 6 | programm | 66 | a |  |  |  |
| 7 | int | 67 | a | r |  |  |

Вывод:

В ходе выполненной лабораторной работы была доработана грамматика языка по варианту курсовой работы так, чтобы была возможность написать несколько тестовых программ. Так же изучены основные идеи и понятия нисходящих методов синтаксического анализа, свойства формальных грамматик, необходимые для реализации нисходящего восстановления дерева грамматического разбора. Приобретены навыки построения процедурной и различных автоматных реализаций нисходящего анализа, исследовано поведение нисходящих синтаксических акцепторов.