МИНИСТЕРСТВО ВЫСШЕГО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

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

Кафедра вычислительной техники



**Лабораторная работа №4**

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

**на тему**: «Синтаксис языков программирования. Нисходящий синтаксический анализ»

Вариант: 11213131

Выполнил:Проверил:

студент гр. АВТ-709, АВТФ *доцент каф. ВТ*

*Антонов К. О. Малявко А. А.*

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

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

(подпись) (подпись)

Новосибирск

2020

Оглавление

[Цель работы 3](#_Toc38547645)

[Задание 3](#_Toc38547646)

[Обновления в описании языка 4](#_Toc38547647)

[Описание грамматики 4](#_Toc38547648)

[Реализация грамматики языка 6](#_Toc38547649)

[Текст процедурной реализации и управляющие таблицы автоматных реализаций нисходящего синтаксического акцептора 7](#_Toc38547650)

[Истории работы процедурной и автоматных реализаций нисходящего синтаксического акцептора 8](#_Toc38547651)

[Вывод 13](#_Toc38547652)

[Список литературы 14](#_Toc38547653)

[Приложение 15](#_Toc38547654)

# 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.

# 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.

# Обновления в описании языка

* изменен синтаксис для включения файлов в проект:

*include* файл

* из языка удалены инкременты/декременты (вместо них используется выражение *+= 1* / *-= 1*);
* теперь можно объявлять функции внутри функций;
* теперь вывод в консоль реализован стандартной функцией *F\_print*(параметры), а пользовательский ввод – при помощи стандартной функции *F\_input*(параметры).
* изменен синтаксис условного оператора:

*when* (выражение) *then {* оператор/блок *}*

***otherwhen*** (выражение) *then {* оператор/блок *}*

*other* *{* оператор/блок *}*

# 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. [1]

# Реализация грамматики языка

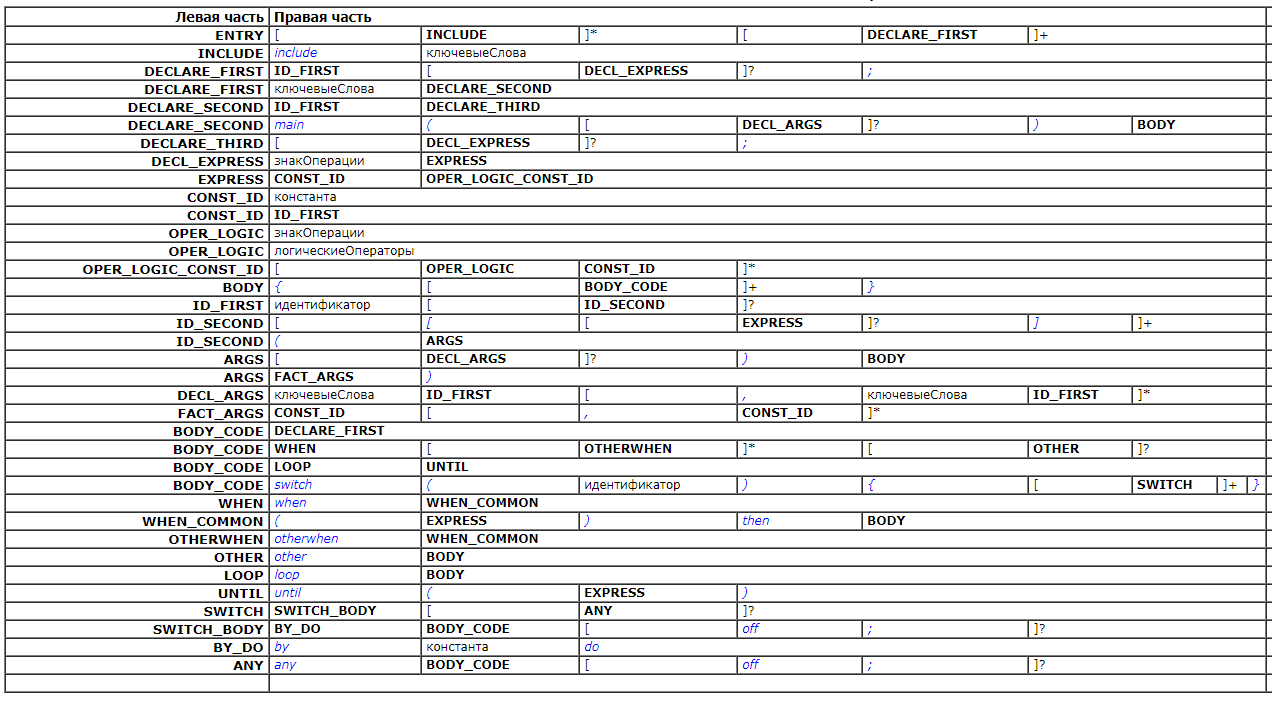


Рисунок 1 – Грамматика языка (файл 4lab\_v13.xml)



Рисунок 2 – Принадлежность грамматики к классу LL(1)

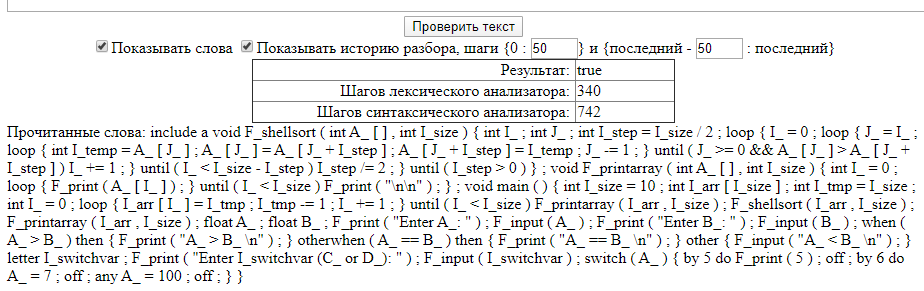


Рисунок 3 – Результат работы лексического и синтаксического анализаторов

Код тестовой программы в приложении.

# Текст процедурной реализации и управляющие таблицы автоматных реализаций нисходящего синтаксического акцептора

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. [1]

Построим таблицу автоматных реализаций.

Для этого выберем шаблон lexAsTableSyntAsSingleFSM и показываем управляющую таблицу нисходящего автомата с одним состоянием. Таблица не помещается в отчет, поэтому приложу её файлом Excel (Лист1).

Далее выбираем шаблон lexAsTableSyntAsMultiFSM и показываем управляющую таблицу нисходящего автомата с несколькими состояниями. Смотрите в файле Excel (Лист2).

# Истории работы процедурной и автоматных реализаций нисходящего синтаксического акцептора

Выберем шаблон lexAsTableSyntAsRD и запустим построенный транслятор. Напишем небольшую правильную с точки зрения грамматики программу:

include a

void main() {

int A\_ = 1;

}

Таблица 1 – История разбора правильной программы (1)

|  |  |  |
| --- | --- | --- |
| 0 | include | ENTRY |
| 1 | include | ENTRY\_0 |
| 2 | include | INCLUDE |
| 3 | void | ENTRY\_0 |
| 4 | void | ENTRY\_1 |
| 5 | void | DECLARE\_FIRST |
| 6 | main | DECLARE\_SECOND |
| 7 | ) | DECLARE\_SECOND\_0 |
| 8 | { | BODY |
| 9 | int | BODY\_0 |
| 10 | int | BODY\_CODE |
| 11 | int | DECLARE\_FIRST |
| 12 | A\_ | DECLARE\_SECOND |
| 13 | A\_ | ID\_FIRST |
| 14 | = | ID\_FIRST\_0 |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
| 15 | = | DECLARE\_THIRD |
| 16 | = | DECLARE\_THIRD\_0 |
| 17 | = | DECL\_EXPRESS |
| 18 | 1 | EXPRESS |
| 19 | 1 | CONST\_ID |
| 20 | ; | OPER\_LOGIC\_CONST\_ID |
| 21 | } | BODY\_1 |
| 22 |  | ENTRY\_2 |

Напишем неправильную с точки зрения грамматики программу:

include a

void main() {

int A = 1;

}

Таблица 2 – История разбора неправильной программы (1)

|  |  |  |
| --- | --- | --- |
| 0 | include | ENTRY |
| 1 | include | ENTRY\_0 |
| 2 | include | INCLUDE |
| 3 | void | ENTRY\_0 |
| 4 | void | ENTRY\_1 |
| 5 | void | DECLARE\_FIRST |
| 6 | main | DECLARE\_SECOND |
| 7 | ) | DECLARE\_SECOND\_0 |
| 8 | { | BODY |

Продолжение таблицы 2

|  |  |  |
| --- | --- | --- |
| 9 | int | BODY\_0 |
| 10 | int | BODY\_CODE |
| 11 | int | DECLARE\_FIRST |
| 12 | A | DECLARE\_SECOND |

Теперь выберем шаблон lexAsTableSyntAsMultiFSM и запустим построенный транслятор. Напишем небольшую правильную с точки зрения грамматики программу:

include a

void main() {

int A = 1;

}

Таблица 3 – История разбора правильной программы (2)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | include | 0 |  |  |  |  |
| 1 | include | 81 |  |  | s |  |
| 2 | include | 37 |  |  |  | e |
| 3 | include | 170 |  |  | s |  |
| 4 | include | 1 |  |  |  |  |
| 5 | include | 83 | a |  |  |  |
| 6 | a | 84 | a | r |  |  |
| 7 | void | 171 |  |  |  |  |
| 8 | void | 37 |  |  |  | e |
| 9 | void | 38 |  |  |  |  |
| 10 | void | 172 |  | r |  |  |
| 11 | void | 82 |  |  |  |  |
| 12 | void | 39 |  |  |  |  |
| 13 | void | 173 |  |  | s |  |
| 14 | void | 2 |  |  |  | e |
| 15 | void | 3 |  |  |  |  |
| 16 | void | 88 | a |  |  |  |
| 17 | main | 89 |  |  |  |  |
| 18 | main | 4 |  |  |  | e |
| 19 | main | 5 |  |  |  |  |
| 20 | main | 92 | a |  |  |  |

Продолжение таблицы 3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 21 | ( | 93 | a |  |  |  |
| 22 | ) | 94 |  |  | s |  |
| 23 | ) | 44 |  |  |  | e |
| 24 | ) | 45 |  |  |  |  |
| 25 | ) | 181 |  | r |  |  |
| 26 | ) | 95 | a |  |  |  |
| 27 | { | 96 |  |  |  |  |
| 28 | { | 15 |  |  |  |  |
| 29 | { | 111 | a |  |  |  |
| 30 | int | 112 |  |  | s |  |
| 31 | int | 50 |  |  |  |  |
| 32 | int | 188 |  |  | s |  |
| 33 | int | 23 |  |  |  | e |
| 34 | int | 129 |  |  |  |  |
| 35 | int | 2 |  |  |  | e |
| 36 | int | 3 |  |  |  |  |
| 37 | int | 88 | a |  |  |  |
| 38 | A\_ | 89 |  |  |  |  |
| 39 | A\_ | 4 |  |  |  | e |
| 40 | A\_ | 90 |  |  | s |  |
| 41 | A\_ | 16 |  |  |  |  |
| 42 | A\_ | 114 | a |  |  |  |
| 43 | = | 115 |  |  |  |  |
| 44 | = | 53 |  |  |  | e |
| 45 | = | 54 |  |  |  |  |
| 46 | = | 194 |  | r |  |  |
| 47 | = | 91 |  |  |  |  |
| 48 | = | 6 |  |  |  |  |
| 49 | = | 97 |  |  | s |  |
| 50 | = | 46 |  |  |  | e |
| 51 | = | 182 |  |  |  |  |
| 52 | = | 7 |  |  |  |  |
| 53 | = | 99 | a |  |  |  |
| 54 | 1 | 100 |  |  |  |  |
| 55 | 1 | 8 |  |  |  |  |
| 56 | 1 | 101 |  |  | s |  |
| 57 | 1 | 9 |  |  |  | e |

Продолжение таблицы 3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 58 | 1 | 103 | a | r |  |  |
| 59 | ; | 102 |  |  |  |  |
| 60 | ; | 13 |  |  |  | e |
| 61 | ; | 14 |  |  |  |  |
| 62 | ; | 110 |  | r |  |  |
| 63 | ; | 98 | a | r |  |  |
| 64 | } | 189 |  |  |  |  |
| 65 | } | 51 |  |  |  | e |
| 66 | } | 52 |  |  |  |  |
| 67 | } | 192 |  | r |  |  |
| 68 | } | 113 | a | r |  |  |
| 69 |  | 174 |  |  |  |  |
| 70 |  | 40 |  |  |  | e |
| 71 |  | 41 |  |  |  |  |
| 72 |  | 177 |  | r |  |  |

Напишем неправильную с точки зрения грамматики программу:

include a

void main() {

int A = 1;

}

Таблица 4 – История разбора неправильной программы (1)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | include | 0 |  |  |  |  |
| 1 | include | 81 |  |  | s |  |
| 2 | include | 37 |  |  |  | e |
| 3 | include | 170 |  |  | s |  |
| 4 | include | 1 |  |  |  |  |
| 5 | include | 83 | a |  |  |  |
| 6 | a | 84 | a | r |  |  |
| 7 | void | 171 |  |  |  |  |
| 8 | void | 37 |  |  |  | e |
| 9 | void | 38 |  |  |  |  |
| 10 | void | 172 |  | r |  |  |
| 11 | void | 82 |  |  |  |  |
| 12 | void | 39 |  |  |  |  |

Продолжение таблицы 4

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 13 | void | 173 |  |  | s |  |
| 14 | void | 2 |  |  |  | e |
| 15 | void | 3 |  |  |  |  |
| 16 | void | 88 | a |  |  |  |
| 17 | main | 89 |  |  |  |  |
| 18 | main | 4 |  |  |  | e |
| 19 | main | 5 |  |  |  |  |
| 20 | main | 92 | a |  |  |  |
| 21 | ( | 93 | a |  |  |  |
| 22 | ) | 94 |  |  | s |  |
| 23 | ) | 44 |  |  |  | e |
| 24 | ) | 45 |  |  |  |  |
| 25 | ) | 181 |  | r |  |  |
| 26 | ) | 95 | a |  |  |  |
| 27 | { | 96 |  |  |  |  |
| 28 | { | 15 |  |  |  |  |
| 29 | { | 111 | a |  |  |  |
| 30 | int | 112 |  |  | s |  |
| 31 | int | 50 |  |  |  |  |
| 32 | int | 188 |  |  | s |  |
| 33 | int | 23 |  |  |  | e |
| 34 | int | 129 |  |  |  |  |
| 35 | int | 2 |  |  |  | e |
| 36 | int | 3 |  |  |  |  |
| 37 | int | 88 | a |  |  |  |

# Вывод

В ходе выполнения лабораторной работы была исправлена и дополнена грамматика языка по варианту курсовой работы. Были изучены основные 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.

# Список литературы

1. Формальные языки и компиляторы : учебник / А. А. Малявко. – Новосибирск: Изд-во НГТУ, 2013. – 431 с. – (Серия «Учебники НГТУ»).

# Приложение

Код тестового примера

include a

void F\_shellsort(int A\_[], int I\_size)

{

int I\_;

int J\_;

int I\_step = I\_size / 2;

loop

{

I\_ = 0;

loop

{

J\_ = I\_;

loop

{

int I\_temp = A\_[J\_];

A\_[J\_] = A\_[J\_ + I\_step];

A\_[J\_ + I\_step] = I\_temp;

J\_ -= 1;

}

until(J\_ >= 0 && A\_[J\_] > A\_[J\_ + I\_step])

I\_ += 1;

}

until(I\_ < I\_size - I\_step)

I\_step /= 2;

}

until(I\_step > 0)

};

void F\_printarray(int A\_[], int I\_size)

{

int I\_ = 0;

loop

{

F\_print(A\_[I\_]);

}

until(I\_ < I\_size)

F\_print("\n\n");

};

void main() {

int I\_size = 10;

int I\_arr[I\_size];

int I\_tmp = I\_size;

int I\_ = 0;

loop

{

I\_arr[I\_] = I\_tmp;

I\_tmp -= 1;

I\_ += 1;

}

until(I\_ < I\_size)

F\_printarray(I\_arr, I\_size);

F\_shellsort(I\_arr, I\_size);

F\_printarray(I\_arr, I\_size);

float A\_;

float B\_;

F\_print("Enter A\_: ");

F\_input(A\_);

F\_print("Enter B\_: ");

F\_input(B\_);

when (A\_ > B\_) then {

F\_print("A\_ > B\_ \n");

}

otherwhen (A\_ == B\_) then {

F\_print("A\_ == B\_ \n");

}

other {

F\_input("A\_ < B\_ \n");

}

letter I\_switchvar;

F\_print("Enter I\_switchvar (C\_ or D\_): ");

F\_input(I\_switchvar);

switch(A\_) {

by 5 do

F\_print(5);

off;

by 6 do

A\_ = 7;

off;

any

A\_ = 100;

off;

}

}