Лабораторная работа №4  
«Синтаксис языков программирования. Нисходящий синтаксический анализ»  
по дисциплине «Теория формальных языков и компиляторов»

|  |  |
| --- | --- |
| Факультет: АВТФ | Вариант: 42243344 |
| Группа: АВТ-815 | Преподаватель: Малявко А.А. |
| Студент: Иванов И.И. |  |
|  |  |

Оглавление

[Цели работы. 3](#_Toc69829817)

[Задание к работе. 3](#_Toc69829818)

[Исходные данные варианта 3](#_Toc69829819)

[Документация к языку программирования 5](#_Toc69829820)

[1. Лексика языка 5](#_Toc69829821)

[2. Синтаксис языка 7](#_Toc69829822)

[3. Семантика языка и технические особенности 7](#_Toc69829823)

[3.1. Приоритеты операций 7](#_Toc69829824)

[3.2. Оператор выбора 8](#_Toc69829825)

[3.3. Входная точка 8](#_Toc69829826)

[3.4. Файлы 8](#_Toc69829827)

[Ход работы. 9](#_Toc69829828)

[1. Расширение грамматики языка из курсовой работы 9](#_Toc69829829)

[2. Проверка грамматики на принадлежность к классу LL1 11](#_Toc69829830)

[3. Построение и изучение различных синтаксических акцепторов 11](#_Toc69829831)

[3.1. Изучение процедурного акцептора 12](#_Toc69829832)

[3.2. Изучение акцептора, основанного на конечном автомате с одним состоянием 13](#_Toc69829833)

[3.3. Изучение акцептора, основанного на конечном автомате с несколькими состояниями 13](#_Toc69829834)

[Вывод. 15](#_Toc69829835)

# Цели работы.

Данная лабораторная работа ставит перед собой несколько целей:

1. Изучение основных идей и понятий нисходящих методов синтаксического анализа;
2. Выявление свойств формальных грамматик, необходимых для реализации нисходящего восстановления дерева грамматического разбора;
3. Приобретение навыков построения процедурной и различных автоматных реализаций нисходящего анализа;
4. Исследование поведения нисходящих синтаксических акцепторов.

# Задание к работе.

Порядок выполнения работы:

1. Расширить грамматику заданного на курсовую работу языка до полной;
2. Изучить и освоить проверку принадлежности грамматики к классу **LL1**, используя в качестве проверяемых грамматики, полученные при выполнении работы №4;
3. Освоить технологию удаления символов из множеств выбора правил с использованием тега <exclude> для приведения грамматики к классу **LL1** и примеров 6IfElseWithConflict и 7IfElseNoConflict;
4. Построить конечный автомат со стековой памятью и несколькими состояниями (шаблон …SyntAsMultiFSM…), разобраться в структуре управляющей таблицы автомата, уяснить способы формирования и использования всех полей;
5. 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;
6. Построить процедурную реализацию рекурсивного спуска (шаблон …SyntAsRD…), уяснить способы формирования функций этого акцептора.
7. 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.
8. 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.
9. Подготовить, сдать и защитить отчет к лабораторной работе.

# Исходные данные варианта

Правила языка из курсовой работы:

1. Лексика
   1. Идентификаторы:
      1. <бБ><пЦ><бБ> (например: x123y, N1N, a0b, …)
   2. Константы:
      1. целые по основанию 10 (int); вещественные и экспоненциальные; символьные (char)
2. Синтаксис
   1. Оператор присваивания:
      1. put<В> to <И> ;
   2. Условный оператор:
      1. at <ЛВ> do <ОБ> [ else do <ОБ> ]
   3. Оператор цикла:
      1. while ( <ЛВ> ) do <ОБ> (внутри ОБ может появиться break;)
   4. Оператор переключателя:
      1. select <В> case ( <К> ) <ОБ> [ break; ] [ case … ] … [ case () <ОБ> ] end
3. Семантика
   1. Примитивные типы данных (целое, вещественное, символьное):
      1. long[\_u], float, string
   2. Формат псевдокода:
      1. Триады: <Код><Оп><Оп>

# Документация к языку программирования

## Лексика языка

1. Идентификаторы. Идентификаторы в данном языке представляют из себя последовательность цифр, заключённых в **открывающую** и **закрывающую** буквы латинского алфавита любого регистра;
2. Специальные символы;
   1. Разграничители:
      1. Символ «,»;
      2. Символ «;».
   2. Скобки:
      1. Квадратные []. Нужны для доступа к конкретной ячейки массива;
      2. Круглые (). Нужны для группировки логических условий;
      3. Фигурные {}. Нужны для группировки операций.
   3. Форматирующие — символы, используемые людьми для упрощения чтения текста и не отображающиеся в явном виде:
      1. \t — табуляция;
      2. \n — переход на новую строку;
      3. \r — переход каретки на начало текущей строки;
      4. Символ пробела.
3. Константы:
   1. Символьные (char). Формат: любая буква латинского алфавита любого регистра, заключённая в одинарные кавычки (‘);
   2. Вещественные (float). Формат: одна цифра; точка, разделяющая целую и дробную части числа; последовательность цифр;
   3. Целые по основанию 10 (int). Формат: последовательность цифр;
   4. Экспоненциальные (float). Формат: конструкция, идентичная вещественному числу; обязательные «+» или «-», обозначающие знак степени числа 10; последовательность цифр, задающая модуль степени числа 10.
4. Ключевые слова. Представляют из себя любые последовательности латинских букв. Необходимы в языке для формирования различных конструкций, например, переходы по условию;
5. Комментарии. Комментарии представляют из себя некий текст, написанный программистом, который не влияет на саму программу и используется людьми как инструмент, помогающий корректно интерпретировать код. Реализовано два типа комментариев:
   1. Однострочные. Формат: //, любая последовательность символов;
   2. Многострочные. Формат: /\* (открывающие символы), произвольная последовательность символов, \*/ (закрывающие символы).
6. Операторы представляют из себя некоторую операцию, которая выполняется над операндами и возвращающая вычисленное значение — результат выполнения операции. Операторы различают по количеству задействованных операндов, а именно — унарные и бинарные:
   1. Унарные:
      1. ++ — присвоить переменной её текущее значение, увеличенное на 1;
      2. -- — присвоить переменной её текущее значение, уменьшенное на 1;
      3. ! — логическая операция НЕ.
   2. Бинарные:
      1. <;
      2. >;
      3. <= — меньше или равно;
      4. >= — больше или равно;
      5. != — не равно;
      6. == — знак равенства чего-то из левой части чему-то из правой части;
      7. && — логическое И;
      8. || — логическое ИЛИ;
      9. –;
      10. +;
      11. /;
      12. \*.

Лексика, записанная на языке регулярных выражений:

Таблица 1 — регулярные выражения, описывающие лексику

| **Имя автомата** | **Имя группы слов** | **Регулярное выражение** | **Действие** | **Примечание** |
| --- | --- | --- | --- | --- |
| main | **Binary operators** | [<>] | ([<][=]) | ([>][=]) | ([!][=]) | ([=][=]) |  |  |
| main | **Binary operators** | ([&][&]) | ("||") |  |  |
| main | **Binary operators** | [-+/\*] |  |  |
| main | **Unary operators** | ([+][+]) | ([-][-]) | [!] |  |  |
| main | **brackets** | [()] | ("\[") | ("\]") | [{}] |  |  |
| main | **const** | [0-9]+ |  | int |
| main | **const** | [0-9]+[.][0-9]+ |  | Вещественное число |
| main | **const** | ['][]['] |  | Символ |
| main | **const** | [1-9][.][0-9]+[e][+-][0-9]+ |  | Экспоненциальная форма записи числа |
| main | **delimiters** | [;] | [,] |  |  |
| main | **formatting word** | [ \t\r\n]+ | ignoreLastWord=true; |  |
| main | **identifier** | [a-zA-Z][0-9]+[a-zA-Z] |  | Буква, НЕПУСТАЯ последовательность цифр, Буква |
| main | **keyword** | [a-z]+ |  |  |
| main | **comments** | [/][/][]\*[\n] | ignoreLastWord=true; | Заметки, создаваемые программистом и не влияющие на саму программу |
| main | **comments** | [/][\*][]\*[\*][/] | ignoreLastWord=true; |  |

## Синтаксис языка

Синтаксис языка описывается таблицей:

Таблица 2 — синтаксис языка

| **Конструкция** | **Структура** |
| --- | --- |
| Комментарий до конца строки | // |
| Присваивание значения переменной | put <значение>to <переменная> |
| Объявление переменной | <тип> <имя> |
| Объявление переменной с присваиванием значения | put <значение>to <тип> <переменная> |
| Группировка выражений | ( ... ) |
| Блок действий | { ... } |
| Определение функции | <тип\_возвращаемого\_значения> <имя\_функции> (<аргумент1>, ..., <аргументN>) |
| Вызов функции | <имя\_функции> (<аргумент1>, ..., <аргументN>) |
| Разделитель команд | ; или конец строки |
| Если - то | at <условие> do <действие> |
| Если - то - иначе | at <условие> do <действие\_если\_истина> else do <действие\_если\_ложь> |
| Цикл с предусловием | while (<условие>) do <тело\_цикла> |
| Прерывание цикла или оператора | break |
| Оператор выбора | select <переменная или выражение> case ( <константа> ) <действие> [ case () <ОБ> ] end (нет ограничений на количество блоков case) |

## Семантика языка и технические особенности

### Приоритеты операций

Порядок арифметических операций определён по принципу, аналогичному тому, что есть в арифметике:

* Действие в ();
* Умножение;
* Деление;
* Сложение и вычитание.

### Оператор выбора

Принцип работы оператора выбора таков:

1. Происходит сопоставление значения, найденного в поле <переменная или выражение> с константами из блоков case ();
2. Если такой блок есть, то начинается выполнение описанных в нём действий, иначе идём на следующий шаг;
3. Выполнение действий во всех последующих блоках до достижения ключевого слова end.

Если необходимо выполнить при совпадении условия только действия из 1 блока case, то тогда в конце каждого из них необходимо добавить ключевое слово break;

### Входная точка

Входной точкой данной программы будет функция main, которая не явно будет вызываться при исполнении.

### Файлы

Язык не поддерживает использование в данном файле других текстовых документов, содержащих код, написанный на этом же языке.

# Ход работы.

## Расширение грамматики языка из курсовой работы

В прошлой лабораторной работе были построены разрозненные грамматики для каждого вида синтаксической конструкции языка: присвоение, условный оператор, оператор цикла. Теперь данные грамматики объединены в одну (файл MyLanguage with Syntax 2.xml):

Таблица — грамматика языка из курсовой работы

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Левая часть** | **Правая часть** | | | | | | **Примечание** |
| **S** | **Op** | | | | | |  |
| **Op** | **Cond** | *do* | **Act** | **R** | | | Условный оператор |
| **Op** |  | | | | | |  |
| **Op** | *put* | **R-value** | *to* | **L-value** | *;* | **Op** | Оператор присвоения |
| **Op** | *while* | *(* | **LogExpr** | *)* | *do* | **RWhile** |  |
| **Cond** | *at* | **LogExpr** | | | | | условие |
| **Cond** |  | | | | | |  |
| **Act** | identifier | Binary\_operators | identifier | *;* | | | действие |
| **R** | *else* | **Op** | | | | |  |
| **R** | **Act** | | | | | |  |
| **R** | *endat* | **Op** | | | | |  |
| **R-value** | const | | | | | | "Правое-значение". Ведет к выражению, но пока что только к константе |
| **L-value** | identifier | | | | | | "Левое-значение". Ведет к идентификатору (т.е. операнду) |
| **LogExpr** | identifier | Binary\_operators | identifier | | | |  |
| **RWhile** | **Op** | | | | | |  |
| **RWhile** | **Act** | **Op** | | | | |  |
|  |  | | | | | |  |

Пример работы:

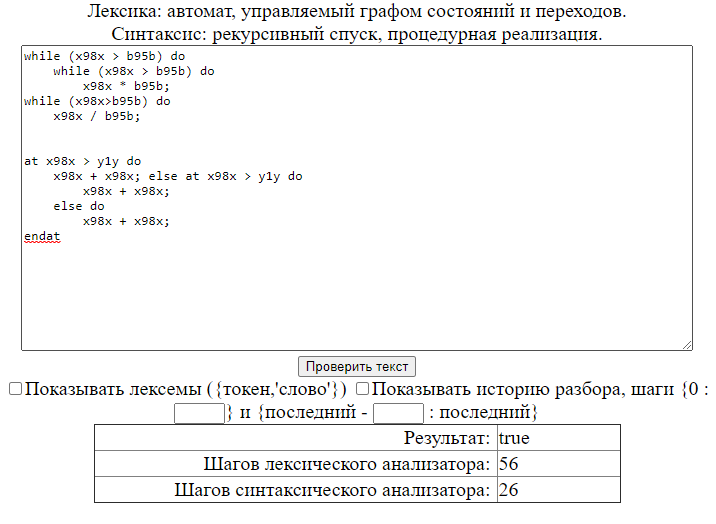


Рисунок — тестирование грамматиики языка из курсовой работы

Код из примера:

while (x98x > b95b) do

while (x98x > b95b) do

x98x \* b95b;

while (x98x>b95b) do

x98x / b95b;

at x98x > y1y do

x98x + x98x; else at x98x > y1y do

x98x + x98x;

else do

x98x + x98x;

endat

Стоит оговориться, что данная грамматика обладает следующими особенностями:

* Не является исчерпывающей, т.е. Не описывает полный набор возможностей, доступных в распространённых сегодня языках программирования;
* Использует заглушки в местах, где, с точки зрения синтаксиса, может присутствовать любой оператор;
* При взгляде на структуру данной синтаксической программы обладает рядом несовершенств.

По этим причинам грамматика языка будет дополняться в последующих работах.

## Проверка грамматики на принадлежность к классу LL1

Перед дальнейшей работой и анализом необходимо ответить на вопросы:

* Что такое LL1-грамматика?
* Как формально определяются критерии выбора того или иного правила грамматики?

К классу LL1-грамматик относятся контекстно-свободные грамматики, обладающие рядом особенностей:

* Если для некоторого нетерминала есть несколько правил то:
  + их правые части начинаются с различных терминальных символов;
  + их множества выбора попарно не пересекаются.
* Есть правила с пустой правой частью.

Каждое правило формальной грамматики обладает так называемым Множеством выбора правила — такое множество, которое состоит терминалов, выводимых прямо или опосредованно для данного правила. Существует несколько формальных способов формирования множеств выбора:

* Если в правой части (ПЧ) **пустота**, то множество выбора состоит из **последователей левой части (ЛЧ)**;
* Если ПЧ **непустая** и содержит хотя бы один **терминал или неаннулируемый нетерминал**, то множество выбора состоит из **множества предшественников ПЧ**;
* Если ПЧ **непустая** и состоит **только из аннулируемых нетерминалов**, то множество выбора является объединением **множества предшественников ПЧ** **и множества последователей ЛЧ**.

Соответственно, для ответа на вопрос о принадлежности грамматики к классу LL1 достаточно построить для каждого правила множество выбора.

## Построение и изучение различных синтаксических акцепторов

В данном пункте и далее работа будет проходить с грамматикой оператора присвоения, являющейся представителем класса LL1 (Таблица 5), (файл MyLanguageInit.xml) ввиду её простоты.

Таблица — грамматика оператора присвоения

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Левая часть** | **Правая часть** | | | | | **Примечание** |
| **S** | **X** | **S** | | | | ведет к присваиванию |
| **S** |  | | | | |  |
| **L** | identifier | | | | | "Левое-значение". Ведт к идентификатору (т.е. операнду) |
| **R** | const | | | | | "Правое-значение". Ведт к выражению, но пока что только к константе |
| **X** | *put* | **R** | *to* | **L** | *;* |  |

Таблица — множества выбора правил грамматики присвоения

|  |  |  |
| --- | --- | --- |
| Правило № | Для нетерминала | Множество выбора: |
| 0 | **S** | *put* |
| 1 | **S** | EOF |
| 2 | **L** | identifier |
| 3 | **R** | const |
| 4 | **X** | *put* |

Методика изучения акцепторов:

* Построение правильного предложения:”put 1.6e-19 to x98x;”;
* Построение неправильного предложения:”put 1.6e-19 in x98x;”;
* Ручная трассировка шагов акцептора для правильного предложения;
* Повторение предыдущего пункта для неправильного предложения.

### Изучение процедурного акцептора

Таблица — история работы процедурного синтаксического акцептора. Правильное предложение

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Такт | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Вход | put | 1.6e-19 | 1.6e-19 | to | x98x | x98x | ; | EOF | EOF | EOF |
| Функция | S | X | R | X | X | L | X | S | S | S |
| Стек |  | S | X | S | S | X | S |  | S |  |
|  |  | S |  |  | S |  |  |  |  |

Таблица — история работы процедурного синтаксического акцептора. Неправильное предложение

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Такт | 0 | 1 | 2 | 3 |
| Вход | put | 1.6e-19 | 1.6e-19 | in |
| Функция | S | X | R | X |
| Стек |  | S | X | S |
|  |  | S |  |

### Изучение акцептора, основанного на конечном автомате с одним состоянием

Таблица — история работы синтаксического акцептора, основанного на автомате с одним состоянием. Правильное предложение.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Такт | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Вход | put | put | put | 1.6e-19 | 1.6e-19 | to | x98x | x98x | ; | EOF |
| Стек | S | X | put | R | 1.6e-19 | to | L | x98x | ; | EOF |
|  | EOF | S | R | to | to | L | ; | ; | EOF |  |
|  |  | EOF | to | L | L | ; | EOF | EOF |  |  |
|  |  |  | L | ; | ; | EOF |  |  |  |  |
|  |  |  | ; | EOF | EOF |  |  |  |  |  |
|  |  |  | EOF |  |  |  |  |  |  |  |

Таблица — история работы синтаксического акцептора, основанного на автомате с одним состоянием. Неправильное предложение.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Такт | 0 | 1 | 2 | 3 | 4 | 5 |
| Вход | put | put | put | 1.6e-19 | 1.6e-19 | in |
| Стек | S | X | put | R | 1.6e-19 | to |
|  | EOF | S | R | to | to | L |
|  |  | EOF | to | L | L | ; |
|  |  |  | L | ; | ; | EOF |
|  |  |  | ; | EOF | EOF |  |
|  |  |  | EOF |  |  |  |

### Изучение акцептора, основанного на конечном автомате с несколькими состояниями

Таблица — история работы синтаксического акцептора, основанного на автомате с несколькими состояниями. Правильное предложение.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Такт | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Вход | put | put | put | put | 1.6e-19 | 1.6e-19 | 1.6e-19 | to | x98x | x98x | x98x | ; | EOF | EOF | EOF | EOF |
| Состояние | 0 | 5 | 4 | 10 | 11 | 3 | 9 | 12 | 13 | 2 | 8 | 14 | 6 | 0 | 1 | 7 |
| Стек |  |  | 6 | 6 | 6 | 12 | 12 | 6 | 6 | 14 | 14 | 6 |  |  |  |  |
|  |  |  |  |  |  | 6 | 6 |  |  | 6 | 6 |  |  |  |  |  |

Таблица — история работы синтаксического акцептора, основанного на автомате с несколькими состояниями. Неправильное предложение.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Такт | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Вход | put | put | put | put | 1.6e-19 | 1.6e-19 | 1.6e-19 | in |
| Состояние | 0 | 5 | 4 | 10 | 11 | 3 | 9 | 12 |
| Стек |  |  | 6 | 6 | 6 | 12 | 12 | 6 |
|  |  |  |  |  |  | 6 | 6 |  |

# Вывод.

В ходе выполнения данной лабораторной работы были изучены и освоены основные свойства формальных грамматик и принципы работы различных синтаксических акцепторов.

Также было обнаружено, что самым эффективным анализатором с точки зрения количества тактов является процедурный синтаксический акцептор.