МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

факультет програмної інженерії та бізнесу

кафедра інженерії програмного забезпечення

**ЛАБОРАТОРНА РОБОТА №3**

з курсу «**Конструювання ПЗ**»

на тему: «**РЕАЛИЗАЦИЯ СИНТАКСИЧЕСКОГО LL(1)-АНАЛИЗАТОРА**»

Виконав: студент 3 курсу групи № 631п

спеціальності

6.050103 «Програмна інженерія»

(шифр і назва напряму підготовки (спеціальності))

Кузьмич М.І.

(прізвище й ініціали студента)

Прийняв: к.т.н., доц. Волобуєва Л. О.

(посада, науковий ступінь, прізвище й ініціали)

Національна шкала:

Кількість балів:

Оцінка ECTS:

Харків – 2018

**Цель работы:** программирование синтаксического анализатора заданных конструкций с помощью синтаксических таблиц.

**Теоретическое введение**

**Синтаксический LL-анализатор** (**LL parser**) — в информатике нисходящий синтаксический анализатор для некоторого подмножества контекстно-свободных грамматик, известных как LL-грамматики. При этом не все контекстно-свободные грамматики являются LL-грамматиками.

Буквы L в выражении «LL-анализатор» означают, что входная строка анализируется слева направо (left to right), и при этом строится её левосторонний вывод (leftmost derivation).

LL-анализатор называется LL(k)-анализатором, если данный анализатор использует предпросмотр на k токенов (лексем) при разборе входного потока. Грамматика, которая может быть распознана LL(k)-анализатором без возвратов к предыдущим символам, называется LL(k)-грамматикой. Язык, который может быть представлен в виде LL(k)-грамматики, называется LL(k)o-языком. Существуют LL(k+n)-языки, которые не являются LL(k)-языками.

Далее описывается анализатор, в основе которого лежит построение таблиц; альтернативой может служить анализатор, построенный методом рекурсивного спуска, который обычно пишется вручную (хотя существуют и исключения, например, генератор синтаксических анализаторов [ANTLR](https://ru.wikipedia.org/wiki/ANTLR) для LL(\*) грамматик).

LL(1)-грамматики очень распространены, потому что соответствующие им LL-анализаторы просматривают поток только на один символ вперед при принятии решения о том, какое правило грамматики необходимо применить. Языки, основанные на грамматиках с большим значением k, традиционно считались трудными для распознавания, хотя при широком распространении генераторов синтаксических анализаторов, поддерживающих LL(k) грамматики с произвольным k, это замечание уже неактуально.

LL-анализатор называется LL(\*)-анализатором, если нет строгого ограничения для k и анализатор может распознавать язык, если токены принадлежат какому-либо регулярному множеству (например, используя детерминированные конечные автоматы).

Существуют противоречия между так называемой «Европейской школой» построения языков, которая основывается на LL-грамматиках, и «Американской школой», которая предпочитает LR-грамматики. Такие противоречия обусловлены традициями преподавания и деталями описания различных методов и инструментов в конкретных учебниках; кроме того, своё влияние оказал [Н. Вирт](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82,_%D0%9D%D0%B8%D0%BA%D0%BB%D0%B0%D1%83%D1%81) из [ETHZ](https://ru.wikipedia.org/wiki/%D0%A8%D0%B2%D0%B5%D0%B9%D1%86%D0%B0%D1%80%D1%81%D0%BA%D0%B0%D1%8F_%D0%B2%D1%8B%D1%81%D1%88%D0%B0%D1%8F_%D1%82%D0%B5%D1%85%D0%BD%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%88%D0%BA%D0%BE%D0%BB%D0%B0_%D0%A6%D1%8E%D1%80%D0%B8%D1%85%D0%B0), чьи исследования описывают различные методы оптимизации LL(1) распознавателей и компиляторов.

**LL – грамматики**

LL(k) – грамматикой называется КС – грамматика, в которой выбор правила в ходу левостороннего вывода однозначно определяется не более чем k очередными символами входной цепочки, считываемой слева направо.

Название «LL» происходит от двух слов left: левосторонний вывод при чтении слева. Самыми удобными для распознавания являются LL(1) грамматики, в которых выбор направления распознавания однозначно определяется очередным входным символом.

Рекурсивный спуск – это детерминированный метод нисходящего разбора КС – языков, порождаемый LL(1) – грамматиками.

Если в грамматике G существует нетерминал А, для которого , где  - непустая цепочка, грамматика G содержит **левую рекурсию**. Грамматика, содержащая левую рекурсию не может быть LL(1) грамматикой.

**Включение действий в синтаксис**

На основе распознавания структуры входного текста строится и его содержательная обработка, трансляция. Синтаксический анализатор служит основой транслятора, предоставляя возможность выполнить необходимые действия по смысловой (семантической) обработке в нужные моменты в соответствии со структурой входной цепочки.

Встраиваемые в распознаватель действия, предназначенные для выполнения смысловой обработки входного текста, называют **семантическими процедурами.**

Рассмотрим использование СП на примере распознавания целых чисел без знака. Обозначим семантические процедуры через  и  и разметим их на синтаксической диаграмме

Целое:



Процедура  будет выполняться в начале обработки и присваивать : у:=0 некому числу 0. переменная у – это формируемое значение числа.

Процедура  будет добавлять считанный символ (цифру) d к текущему значению у.

: у := 10 \* у + 

**Табличный LL(1) – анализатор**

Рассматривая автоматные языки, мы использовали детерминированный конечный автомат в роли распознавателя. Один из вариантов его реализации – программная интерпретация таблицы переходов автомата. На похожих принципах может быть построен и распознаватель для LL(1) – грамматик.

Вначале модифицируем таблицу переходов КА. Ее обычный формат таков:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Состояние | Символ | | | | |
| а | b | c | d | … |
|  |  |  |  |  |  |
| … |  |  |  |  |  |

Все состояния пронумерованы. Количество допустимых символов может быть большим. Чтобы сохранить размер таблицы, различные символы группируют и каждую группу символов размещают в отдельном столбце.

В каждом состоянии проверяется совпадении с определенным символом.

Для примера возьмем КА, распознающий целые числа без знака.

|  |  |  |
| --- | --- | --- |
| Состояние | Символ | |
| Цифра | не цифра |
| 1 | 2 | Е |
| 2 | 2 | К |
| Е | Е | Е |
| К |  |  |

Следующая таблица имеет модифицированный вид. Символы записываются во втором столбце, состояние в которое переходит автомат при совпадении входного символа и символа в таблице – в третьем.

В четвертом столбце отмечено, возникает ли ошибка при несовпадении символов. Если в графе ошибка записано «-», то при несовпадении символов автомат переходит в следующее по порядку состояние.

|  |  |  |  |
| --- | --- | --- | --- |
| Состояние | Символ | Переход | Ошибка |
| 10 | цифра | 11 | + |
| 11 | цифра | 11 | - |
| 12 | цифра | 0 | - |

Переход в состояние 0 означает завершение работы автомата с принятием соответствующей части входной цепочки.

Далее построим таблицу переходов для автомата, распознающего степень СД:

степень:



Таблица переходов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Состояние | Символ | Переход | Ошибка | Вызов | Читать |
| 20 |  | 22 | - | - | + |
| 21 | Любой | 0 | - | - | - |
| 22 | Любой | 10 | - | + | - |
| 23 | Любой | 0 | - | - | - |

Столбец «читать» управляет чтением следующего символа : если «+», то при совпадении текущего символа с символом во втором столбце читается следующий символ входной цепочки.

«+» в столбце «вызов» означает, что происходит переход с возвратом , вызвав для обработки показателя степени состояние 10, проходит работа автомата, распознающего целые, дойдя до состояния 12 возвращается в состояние 23.

Значение «0» в графе «переход» - возврат в состояние, следующее за тем, из которого произошел вызов.

**Постановка задачи**

1. Для каждой синтаксической диаграммы, построенной в лабораторной работе №1 написать синтаксическую таблицу.
2. На основе построенных таблиц разработать программное обеспечение синтаксического анализатора заданных конструкций. Анализатор должен либо сообщать о том, что конструкция записана верно, либо выдавать сообщение об ошибке с указанием места ее обнаружения.

**Вариант №11**

Уравнение плоскости вида  с целыми коэффициентами.

Описание построенной грамматики

1. – 2 тип
2. – 3 тип
3. – 2 тип
4. – 2 тип
5. – 3 тип
6. – 3 тип

Общая грамматика 2-го типа.

**Синтаксическая таблица для заданной конструкции**

Синтаксическая таблица представлена Таблицей 1.

Таблица 1 – Синтаксическая таблица

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Состояние** | **Символ** | **Переход** | **Ошибка** | **Читать** | **Сообщение ошибки** |
| 1 | – | 2 | – | + | – |
| 2 | Цифра | 3 | + | – | “Ожидался коэффициент” |
| 3 | 0 | 5 | – | + | – |
| 4 | Цифра | 4 | – | + | – |
| 5 | ‘x’ | 6 | + | + | “Ожидался ‘x’” |
| 6 | + | 8 | – | + | – |
| 7 | – | 7 | + | + | “Ожидался ‘+’ или ‘–’” |
| 8 | Цифра | 9 | + | – | – |
| 9 | 0 | 11 | – | + | – |
| 10 | Цифра | 10 | – | + | – |
| 11 | ‘y’ | 12 | + | + | “Ожидался ‘y’” |
| 12 | + | 14 | + | + |  |
| 13 | – | 14 | + | + | “Ожидался ‘+’ или ‘–’” |
| 14 | Цифра | 15 | + | – | – |
| 15 | 0 | 17 | – | + | – |
| 16 | Цифра | 16 | – | + | – |
| 17 | ‘z’ | 18 | + | + | “Ожидался ‘z’” |
| 18 | + | 23 | – | + | – |
| 19 | – | 23 | – | + | – |
| 20 | = | 21 | + | + | “Ожидался ‘=’” |
| 21 | 0 | 22 | + | – | “Ожидался ‘0’” |
| 22 | Любой | 0 | – | – | – |
| 23 | Цифра | 24 | + | – | “Ожидался коэффициент” |
| 24 | 0 | 20 | – | + | – |
| 25 | Цифра | 25 | – | + | – |
| 26 | Любой | 20 | – | – | – |

**Экранные формы работы ПО синтаксического анализатора**

Работа программы представлена на рисунках 17-20.

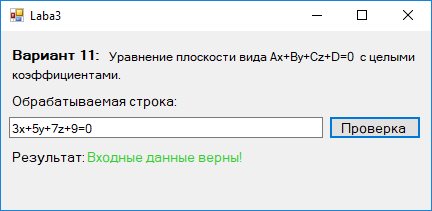


Рисунок 17 – Тестовый пример №1

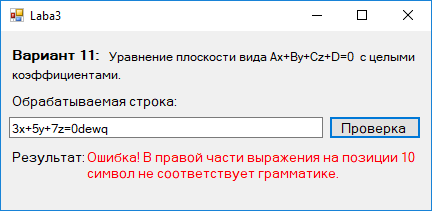


Рисунок 18 – Тестовый пример №2

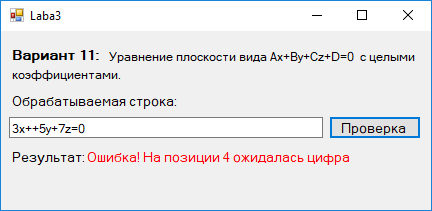


Рисунок 19 – Тестовый пример №3

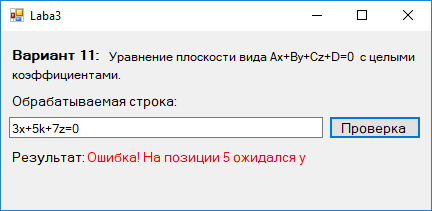


Рисунок 20 – Тестовый пример №4

**Вывод**

В ходе выполнения данной лабораторной работы был разработан синтаксический анализатор конструкций, построенных в лабораторной работе №1 с помощью синтаксических таблиц.

**ПРИЛОЖЕНИЕ А**

Листинг программы

Файл **Row.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConstruirovanieLaba3

{

class Row

{

public int State { get; set; }

public string Symbol { get; set; }

public int Transition { get; set; }

public bool Error { get; set; }

public bool Read { get; set; }

public string Exception { get; set; }

public Row(int State, string Symbol, int Transition, bool Error, bool Read, string Exception)

{

this.State = State;

this.Symbol = Symbol;

this.Transition = Transition;

this.Error = Error;

this.Read = Read;

this.Exception = Exception;

}

}

}

Файл **Table.cs**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace ConstruirovanieLaba3

{

class Table

{

public List<Row> table;

private Label label;

public Table(Label label)

{

table = new List<Row>

{

new Row(1,"-",2,false,true,""),

new Row(2, "Number", 3, true, false,"ожидался коэффициент."),

new Row(3, "0", 5, false, true,""),

new Row(4, "Number", 4, false, true,""),

new Row(5, "x", 6, true, true,"ожидался \"x\"."),

new Row(6, "+", 8, false, true,""),

new Row(7, "-",8,true,true,"ожидался \"+\" или \"-\"."),

new Row(8, "Number", 9, true, false,"ожидался коэффициент."),

new Row(9, "0", 11, false, true,""),

new Row(10, "Number", 10, false, true,""),

new Row(11, "y", 12, true, true,"ожидался \"y\"."),

new Row(12, "+", 14, false, true,""),

new Row(13, "-",14, true, true,"ожидался \"+\" или \"-\"."),

new Row(14, "Number", 15, true, false,"ожидался коэффициент."),

new Row(15, "0", 17, false, true,""),

new Row(16, "Number", 16, false, true,""),

new Row(17, "z", 18, true, true,"ожидался \"z\"."),

new Row(18, "+",23,false,true,""),

new Row(19, "-", 23, false, true,""),

new Row(20, "=", 21, true, true,"ожидалось \"=\"."),

new Row(21, "0", 22, true, false,"ожидался \"0\"."),

new Row(22, "Any", 0, false, false,""),

new Row(23, "Number", 24, true, false,"ожидался коэффициент."),

new Row(24, "0", 20, false, true,""),

new Row(25, "Number", 25, false, true,""),

new Row(26, "Any",20, false, false,"")

};

this.label = label;

this.label.Visible = false;

}

public void Check(string str)

{

str.Trim();

str = str.Replace(" ", "");

str += ";";

int i = 0, j = 0;

while (true)

{

if (table[i].Transition == 0)

{

break;

}

char k = str[j];

switch (table[i].Symbol.Length)

{

case 1:

if (k.ToString() == table[i].Symbol)

{

Processing(ref i, ref j);

continue;

}

if (table[i].Error)

{

ErrorShow(i, j, table[i].Exception);

return;

}

break;

case 3:

i = table[i].Transition - 1;

continue;

case 6:

if (Char.IsDigit(k))

{

Processing(ref i, ref j);

continue;

}

if (table[i].Error)

{

ErrorShow(i, j, table[i].Exception);

return;

}

break;

default:

break;

}

i++;

}

if (j == str.Length - 2)

{

label.Visible = true;

label.ForeColor = Color.LimeGreen;

label.Text = "Входные данные верны!";

}

else

{

label.Visible = true;

label.ForeColor = Color.Red;

label.Text = "Ошибка! Правая часть выражения содержит недопустимый символ на позиции " + (j + 2) + ".";

}

}

private void ErrorShow(int i, int j, string message)

{

label.Visible = true;

label.ForeColor = Color.Red;

label.Text = "Ошибка! На позиции " + (j + 1) + " " + message;

}

private void Processing(ref int i, ref int j)

{

if (table[i].Read)

{

j++;

}

i = table[i].Transition - 1;

}

}

}

Файл **Form1.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace ConstruirovanieLaba3

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void btn\_check\_Click(object sender, EventArgs e)

{

Table t = new Table(label\_result);

t.Check(inp\_str\_text.Text);

}

}

}