Содержание

Введение

Задачей данного курсового проекта является разработка компилятора подмножества языков программирования. Данная система актуальна для студентов высших курсов. Цель - обучения основам программирования, искусства составления программ.

В данной работе использовалась литература Альфреда В. Ахо – Компиляторы: принципы, технологии и инструментарий, 2-е издание. Основным заданием курса являлось написание курсовой работы, выполняющийся группами студентов и состоящей в реализации небольшого языка программирования. В книге представлены такие темы как – лексический анализ, регулярные выражения, методы синтаксического анализа (нисходящие и восходящие). Данная книга включает как теорию, так и упражнения. Книга подготовлена несколькими ведущими университетами страны.

1. Анализ технического задания

Заданием на курсовой проект является разработка транслятора с подмножества языка Алгол. Критерии учитываемые при разработке транслятора: обеспечить развернутую диагностику ошибок программы, реализация отдельного класса транслятора, синтактический разбор выполняется на основе LL(k)-грамматик, разбор сложных выражений методом Дейкстры. В языке поддерживается: не менее 3-х директив описания переменных, сложный арифметический оператор, условный оператор for…step…until…do…, так же представлен пример программы на заданном языке.

В программе поддерживается ввода – вывода данных в текстовом формате с расширением файла «.algol». В программе предусмотрены требования к надежности – несуществующий, пустой, неправильный файл, блокировка некорректных действий пользователя.

2. Описание грамматики языка

В эталонном языке АЛГОЛ используется 116 основных символов. По своему назначению они объединены в 4 основных группы: буквы латинского алфавита строчные и прописные; десять арабских цифр; логические значения булевских переменных; ограничители: знаки арифметических операций, скобки, разделители, описатели, спецификаторы.

Идентификатор – это либо последовательность букв, либо последовательность букв и цифр, причем цифра не может быть первой.

Переменная – это величина, значением которого может быть либо число (тип integer или real), либо логическое значение (тип Boolean).

Служебные слова: begin, end, for, if, else, step, until, do, while, comment, string, label, value, procedure, array.

Грамматика языка программирования содержит правила двух типов: первые (определяющие синтаксические конструкции языка) легко поддаются формальному описанию; вторые (определяющие семантические ограничения языка) обычно излагаются в неформальном виде. Поэтому любое описание языка программирования обычно состоит из двух частей: сначала формально излагаются правила построения синтаксических конструкций, затем на естественном языке дается описание семантических правил. Для компилятора семантические ограничения должны быть представлены в виде алгоритмов проверки правильности программы.

Согласно формуле Бекуса-Наура:

- символ «::=» отделяет левую часть правила от правой;
- нетерминалы обозначаются произвольной символьной строкой, заключенный в угловые скобки «»;
- каждое правило определяет порождение нескольких альтернативных цепочек, отделяемых друг от друга символом «|».

Пример описания идентификатора:

<буква>::=A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

<цифра>::= 0|1|2|3|4|5|6|7|8|9

<идентификатор> ::= <буква>|<идентификатор><буква>|<идентификатор><цифра>

Описание грамматики языка:

<Программа> ::= BEGIN <СписокОператоров> END

<СписокОператоров> ::= <Оператор>|<СписокОператоров><Оператор>

<Оператор> ::= <Объявление>|<Цикл>|<Присваивание>

<Объявление> ::= <Тип><СписокИдентификаторов>;

<Тип> ::= REAL|INTEGER|BOOLEAN

<СписокИдентификаторов> ::= id | id,<СписокИдентификаторов>

<Цикл> ::= FOR id := <Операнд> STEP <Операнд> UNTIL <Операнд> DO BEGIN <СписокОператоров> END

<Операнд> ::= lit | id

<Присваивание> ::= id := <СложноеАрифметическоеВыражение>;

Для того что бы привести таблицу FIRST, в нисходящем анализе нужно устранить левую рекурсию и левую факторизацию. Ниже ход преобразований:

<СписокОператоров> ::= <Оператор>|<СписокОператоров><Оператор>

и

<СписокИдентификаторов> ::= id | id,<СписокИдентификаторов>

результат преобразований выглядит так:

<СписокОператоров> ::= <Оператор><СписокОператоров1>

<СписокОператоров1> ::= <Оператор><СписокОператоров1>|ε

и

<СписокИдентификаторов> ::= id<СписокИдентификаторов1>

<СписокИдентификаторов1> ::= , id |ε

Из результатов выше, отсутствует левая рекурсия и левая факторизация, следует, что грамматика принадлежит нисходящему классу.

3. Разработка архитектуры системы и алгоритмов

В данной системе реализован пакетный интерфейс, методология проектирования выбрана функционально-модульная, структура системы – цельная.

Лексический анализатор – это часть компилятора, которая читает литеры программы и строит из них слова (лексемы) исходного языка. Лексический анализатор выделяет из текста лексемы различных типов: идентификаторы, литералы (числовые и символьные константы), разделители.

Описание алгоритма:

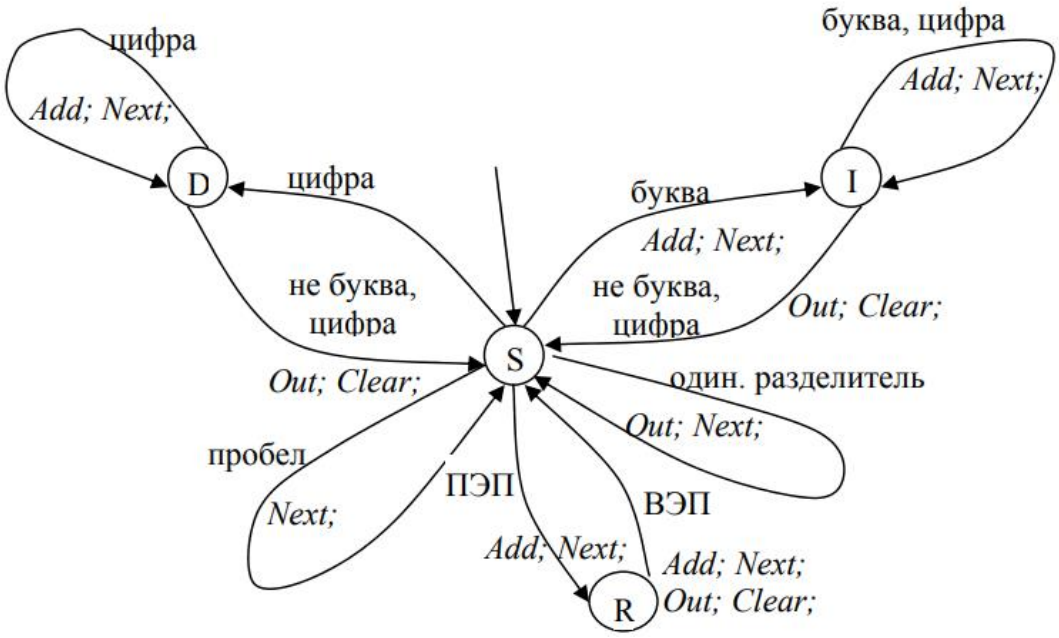

Рис. 1. Пример диаграммы состояний с действиями.

На рисунке 1 приведена диаграмма действий автомата, реализующего лексический анализ.

− Add – добавить очередной символ в конец буфера;

− Next – переместиться к следующему элементу входного потока;

− Out(лексема, тип) – выдать информацию о накопленной лексеме в выходной поток. Тип задаётся веткой диаграммы состояний по которой была собрана лексема;

− Clear – очистить буфер;

− ПЭП и ВЭП – первый и второй элементпары соответственно;

− Все состояния диаграммы – конечные.

Алгоритм разбора цепочек символов по диаграмме состояний с действиями:

- Входные данные лексического анализатора – текст транслируемой программы на входном языке.
- Выходные данные – последовательность лексем (с указанием их предварительного типа).

1. Объявляем текущим начальное состояние S диаграммы.
2. До тех пор, пока не будет достигнуто конечное состояние диаграммы на последнем элементе входного потока или состояние ERROR, считываем очередной символ анализируемой строки и переходим из текущего состояния диаграммы в другое по дуге, помеченной этим символом, выполняя при этом соответствующие действия. Состояние, в которое попадаем, становится текущим.
3. Выходной поток формируется вызовом подпрограммы out.

Алгоритм построения синтаксического анализатора по методу рекурсивного спуска:

1) Каждому нетерминалу в правой части правила соответствует вызов одноимённой подпрограммы, выполняющей анализ цепочек, порождаемых этим нетерминалом;
2) - Каждому терминалу в правой части правила соответствует проверка на совпадение текущего элемента входного потока с терминальным элементом правила.

   - Если элементы совпали, то перемещаемся к следующему элементу и во входном потоке, и в правиле. Если совпадения нет, фиксируем ошибку.
3) Если нетерминалу в левой части правила соответствует несколько альтернативных правых частей, то до выполнения пунктов 1 и 2 надо выбрать правило, по которому далее пойдёт разбор.

Такой выбор выполняется на основе анализа символов входного потока (не более k) и элементов множеств FIRST решающей таблицы.

Таблицу с множествами FIRST:

<Программа>	begin
<СписокОператоров>	real, integer, boolean, id, for
<СписокОператоров1>	real, integer, boolean, id, for, ε
<Оператор>	real, integer, boolean, id, for
<Объявление>	real, integer, boolean
<Тип>	real, integer, boolean
<СписокИдентификаторов>	id,
<СписокИдентификаторов1>	« , », ε
<Цикл>	for
<Операнд>	lit, id
<Присваивание>	id

Разбор сложного арифметического выражения методом Дейкстра.

Метод Дейкстры использует таблицу приоритетов операций и переводит выражение в обратную польскую нотацию.

Таблица приоритетов операций

Операция	**( )**	**+ -**	*** /**
Приоритет	**0**	**1**	**2**

В трансляции участвуют: входная строка (выражение), выходная строка (ОПН), стек операций.

Выражение в примере к курсовому заданию: **c+b*3**

Перевод в обратную польскую нотацию.

Стек	Вход	Выход
ε	c	c
+	+	
	b	cb
+*	*	
	3	cb3
ε	$	cb3*+

Перевод из обратной польской нотации в матричную форму

Стек	Вход	Выход
c	c	
cb	b	
cb3	3	
cM1	*	M1:* 3b
M2	+	M2:+M1c

4. Методика испытаний

Испытания – текст

5. Руководство пользователя

Назначение программы: программа, разработанная в курсовой работе, предназначена для трансляции с подмножества языка Алгол. Выполняет лексический и синтаксический анализ, а также разбор сложного выражения.

Системные требования:

1) ОС Windows 7, 8, 10
2) Процессор: не менее 1 ГГц
3) ОЗУ: 1 ГБ (для 32-разрядных систем) или 2 ГБ (для 64-разрядных систем)
4) Место на жестком диске: 30 Кб;
5) Видеоадаптер: DirectX версии 9 или более поздней;

Для запуска программы достаточно скопировать файл с расширением .exe в любую папку, расположенную на жестком диске и запустить его. Для деинсталляции достаточно лишь удалить файл расширением .exe с диска.

Подготовка к работе:

1. Запустите приложение;
2. В меню выбора выберите пункт «открыть»;
3. Выберите файл с расширением «.algol»;
4. При удачном открытии вы увидите код программы в окне;
5. При неудачном открытии программа выдаст исключение;

Примечание: вы можете написать вручную или изменять его в окне самостоятельно.

Работа с программой:

1. В меню выбора выберите пункт «выполнить»;
2. Программа начнет выполнения анализа кода и сложного выражения;
3. В случае ошибок, программа выдаст исключение и укажет место ошибки;
4. В случае успеха программа сообщит вам об этом;
5. Следом появится окно разбора данных, выполненный при лексическом анализе.

6. В верхнем левом углу можно выбрать окно разбора сложных выражений

После успешной проверки данных вы можете закрыть окно «Таблицы» и приступить к форматированию или добавлению кода в программу.

6. Руководство программиста

Программа написана на языке программирования C#. Назначением является трансляция языка Алгол. В программе выполняется лексический и синтактический анализы, так же включена проверка сложных арифметических выражений.

Программа состоит из двух форм: MainForm и TableForm.

В программе предусмотрены отдельные классы для разных задач.

Класс Program – главный класс начала программы, передающий исполнение в форму MainForm.

Класс Analiz – класс, представляющий обработку текстовых данных. Выполняет лексический анализ, синтаксический анализ и разбор сложных выражений методом Дейкстры. Также содержит код вывода данных на форму TableForm.

Класс ERR – класс, содержащий исключения, возникающие в ходе выполнения анализов и разбора.

В классах содержатся комментарии к переменным.

Использованы типы данных: строковые данные string, массивы строковых данных string[], список объектов лист, символьное представление chat, логические данные bool. коллекция данных stack.

Глобальные переменные в подклассе analiz:

slova – ключевые слова, использованные в подмножестве языка Алгол;

alltype – лист типов;

allsymbol –лист символов;

table1, table2, table3, table4, table5 – листы для хранения данных относящихся к лексическому анализу. таблица ключевых слов, разделителей, литералов, тсс;

Глобальные переменные в подклассе SNX:

symbolnum – переменная хранения номера лексемы;

tablenum1, tablenum2 – следующие по порядку номера за symbolnum;

lexem0, lexem, lexem1, lexem2 – лексемы в строковом представлении и следующие по порядку лексемы.

Глобальные переменные в подклассе logical:

flag – значение да/нет используемое при достижении конца списка/стака;

raz, oper – значение да/нет для ограничения использования подряд идущих разделителей или операндов;

E, T – стаки, используются при разборе выражения по методу Дейкстры;

operandnum- номер операнда в листе символов;

expr – лист символов, содержащий выражение;

arifmet – массив арифметических действий;

matrix – лист для записей формул, по методу Дейкстры;

Элементы, использованные в проекте: form, datagridviewer, richtextbox, label, tabcontrol, menustrip.

Заключение

Данная программа выполнена в

Список используемой литературы

1.

Приложения

Интерфейс программы:

Рисунок 1. Главное окно программы;

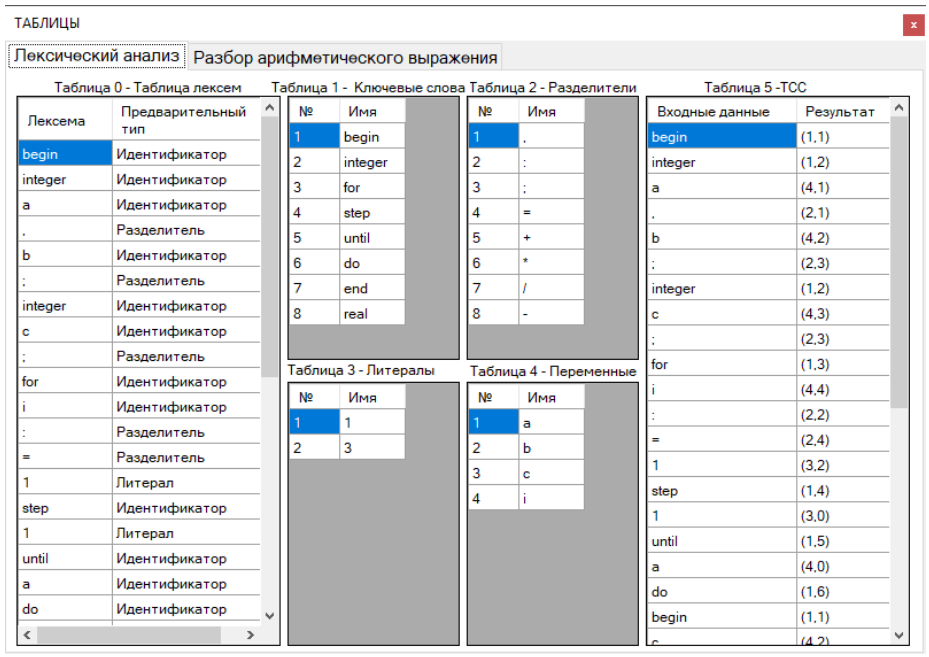

Рисунок 2. Окно программы «Таблицы»;

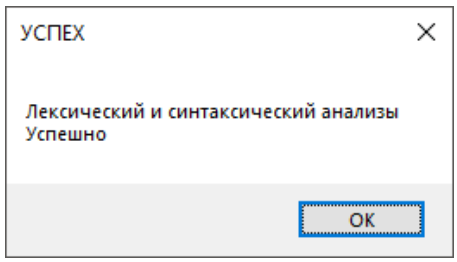

Рисунок 3. Сообщении об успехе проведенного анализа;

Структурная схема программы:

Рисунок 4. Схема.

Выполнение нагрузок на программу описанных в п.4 курсового проекта.

Тест 1.

```
begin integer a,b;
integer c;
for i:= 1 STEP 1 UNTIL a DO BEGIN
c:= c+b*3;
b:=1;
end
end
```

Тест 2.

```
begin integer a,b;
integer c;
integer c2;
integer c3;
integer c4;
integer c5;
integer c6, c7;
integer d1,d2,d3,d4,d5,d6,d7,d8;
for i:= 1 STEP 1 UNTIL a DO BEGIN
c:= c+b*3;
c7:= c2+c3+c4+c5+c6;
d8:=d1*d2*d3*d4*d5*d6*d7;
b:=1;
end
end
```

Тест 3. Намеренная ошибка в типе integer (inter)

```
begin integer a,b;
inter c;
for i:= 1 STEP 1 UNTIL a DO BEGIN
c:= c+b*3;
b:=1;
end
end
```

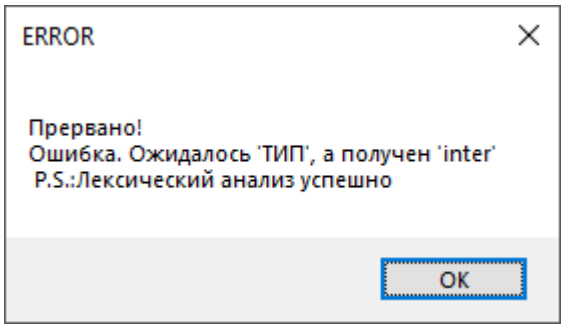

Рисунок 4. Исключения для теста 3.

Тест 4.

```
begin integer a,b;
integer c;
for i:= 1 STEP 1 UNTIL a DO BEGIN
c:= a*b*a+b+c/a+c+a-a*c+a+b+c+b*3;
b:=1;
end
end
```

Тест 5.

а. Намеренная ошибка с добавлением знака арифметики. (+)

```
begin integer a,b;
integer c;
for i:= 1 STEP 1 UNTIL a DO BEGIN
c:= b++c+b*3;
b:=1;
end
end
```

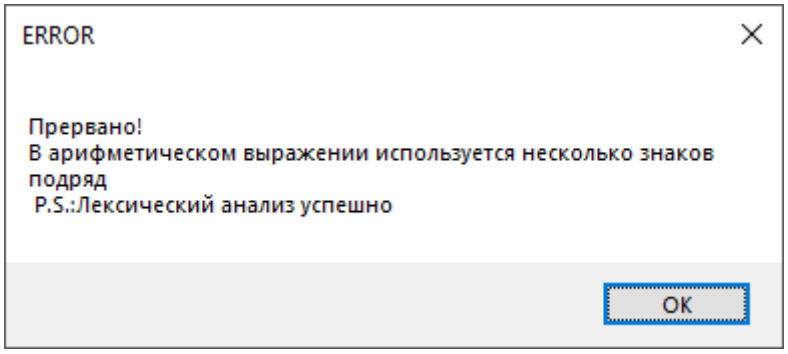

Рисунок 5. Исключение для теста 5.а;

б. Намеренная ошибка с добавлением операнда (c)

```
begin integer a,b;
integer c;
for i:= 1 STEP 1 UNTIL a DO BEGIN
c:= b+ccc+b*3;
b:=1;
end
```

end

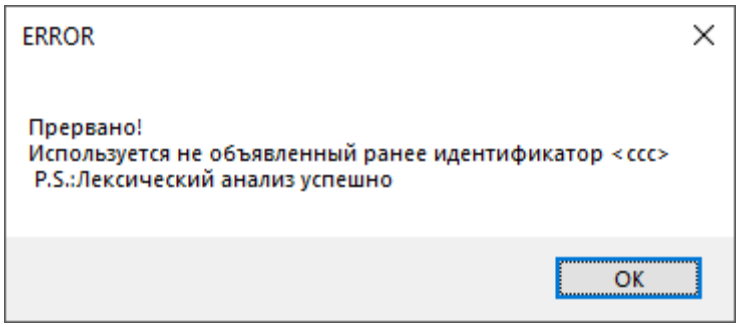

Рисунок 6. Исключение для теста 5.б;