Министерство науки и высшего образования Российской Федерации
Муромский институт (филиал)
Федерально государственного бюджетного образовательного учреждения
высшего образования
«Владимирский государственный университет имени Александра
Григорьевича и Николая Григорьевича Столетовых»

Факультет ИТР
Кафедра ПИн

КУРСОВАЯ РАБОТА

по ТЕОРИЯ АВТОМАТОВ И ФОРМАЛЬНЫХ ЯЗЫКОВ
(наименование дисциплины)

Тема Транслятор с подмножества языка С

Руководитель

_____ Кульков Я.Ю.
(оценка) (фамилия, инициалы)

_____
(подпись) (дата)

Члены комиссии

Студент ПИн-120
(группа)

_____ Андронов И.А.
(оценка) (Ф.И.О.) (фамилия, инициалы)

_____ _____
(подпись) (дата) (подпись) (дата)

_____
(оценка) (Ф.И.О.)

_____
(подпись) (дата)

Муром 2022

В данной курсовой работе был разработан транслятор с подмножества языка C. В ходе выполнения курсовой работы произведен анализ и сбор требований к разрабатываемому транслятору. Синтаксический разбор выполнен на основе LR(k)-грамматик; Разбор выражений выполнен методом Дейкстры. В качестве конструкции языка используется цикл for. Реализован класс транслятора.

In this course work, a translator was developed from a subset of the C language. In the course of the course work, an analysis and collection of requirements for the translator being developed was made. Syntactic parsing is based on LR(k)-grammars; The parsing of expressions was performed by the Dijkstra’s method. The for loop is used as a language construct. Translator class implemented.

Министерство науки и высшего образования Российской Федерации
Муромский институт (филиал)
Федерального государственного бюджетного образовательного учреждения
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

Факультет ИТР

Кафедра ПИн

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

По ТЕОРИЯ АВТОМАТОВ И ФОРМАЛЬНЫХ ЯЗЫКОВ

Тема Транслятор с подмножества языка С

Руководитель

Кульков Я.Ю.
(фамилия, инициалы)

(подпись) (дата)

Студент ПИН - 120
(группа)

Андронов И.А.
(фамилия, инициалы)

(подпись) (дата)

Муром 2022

Содержание

Введение

Появление компьютеров требовало создания языков программирования для взаимодействия с ними. Одним из наиболее широко используемых языков программирования является C, который будет рассмотрен в этой курсовой работе. Для того, чтобы программы были корректно написаны и затем верно распознаны и интерпретированы, используются специальные методы их анализа и преобразования, которые основаны на теории языков и формальных грамматик, а также теории автоматов. Системы, которые используются для анализа и интерпретации программных текстов, называются трансляторами.

Целью курсовой работы является разработка транслятора с подмножества языка C.

Для выполнения цели курсовой работы, были выделены следующие задачи:

– Ознакомление с математическим аппаратом: формальными грамматиками, используемыми для описания искусственных языков;

– Проектирование искусственного языка;

– Формальное описание искусственного языка;

– Отладка лексического и синтаксического анализаторов, входящих в состав проектируемого транслятора;

– Разработка семантических программ транслятора;

– Генерация ассемблерного файла;

– Комплексная отладка транслятора на контрольных (тестовых) примерах.

1. Анализ технического задания

Для начала, нужно понять, что такое транслятор. Транслятор - это программа или инструмент, который выполняет перевод программы из одного языка программирования в другой. Он также выполняет диагностику ошибок, создает словарь идентификаторов и выдает текст программы для печати и т.д.

В соответствии с требованиями к курсовой работе, разрабатываемый транслятор, который поддерживает подмножество языка С, должен иметь следующие характеристики:

- Распознавание 8 символов у идентификатора;
- Не менее 3х директив описания переменных;
- Разбор сложных арифметических операторов;
- Разбор простых логических выражений;
- Оператор цикла for (…;…;…) { … };
- Разбор выражений с помощью метода Дийкстры;

Для реализации транслятора будет выбран современный язык программирования C#, который является объектно-ориентированным и типобезопасным языком и позволяет разработчикам создавать безопасные и надежные приложения, работающие в .NET.

Для создания транслятора будет использована среда разработки Visual Studio Community Edition 2022, которая позволяет создавать не только консольные приложения, но и приложения с графическим интерфейсом, включая поддержку технологии Windows Forms.

2. Описание грамматики языка

2.1 Описание языка

С - это процедурный язык программирования, разработанный в начале 1970-х годов. Он является низкоуровневым языком, что позволяет программистам получить полный контроль над вычислительными процессами, а также использовать высокоэффективные алгоритмы и структуры данных. С помощью Си, можно разрабатывать как системное ПО, так и приложения для персональных компьютеров.

2.2 Описание синтаксиса операторов и основных конструкций

Рассматриваемое подмножество языка си включает в себя следующие операторы: объявление переменных, присвоение значений, арифметический оператор, логический оператор, оператор цикла.

Рассмотрим синтаксис каждого из них:

- Объявление переменных: int a,b;
- Присвоение значений: c=0; b=1; b=5; c=c+b*3;
- Арифметические операторы: c+b*3; b++
- Логический оператор: b<5;
- Оператор цикла: for(b=1; b<5; b++){}

2.3 Разработка грамматики языка

Для описания синтаксиса языка Си была использована форма Бэкуса-Наура. Чтобы построить соответствующую грамматику, нужно следовать следующим шагам:

1. Определяем множество терминальных символов T, которые включают в себя все ключевые слова и разделители языка.
2. Определяем общую структуру программы и начальный символ грамматики, который называется <программа>.
3. Программа в языке Си состоит из одной или более функций, каждая из которых ограничена фигурными скобками. Главной функцией в программе консольного приложения на языке Си является функция main(), которая является основным блоком.
4. Последовательно надо объяснить структуру всех нетерминалов, введенных в первом и в следующих правилах. Сама программа представляет собой множество операторов. Соответственно, необходимо ввести правила, позволяющее описать эту последовательность. Такие правила должны содержать рекурсию, для того чтобы размножить "элемент списка" в количестве одной или более единиц.
5. Если требуется описать часть оператора, которая может принимать различные значения или может отсутствовать, то следует ввести новый нетерминал на месте этой части, а затем в следующем правиле описать ее структуру.
6. Все полученные правила формируют множество P, при этом все нетерминалы, используемые в правых частях правил, должны быть описаны как правила.
7. Набор нетерминалов образует множество N.

После выполнения всех шагов алгоритма была получена следующая грамматика:

G=(T,N,P,<программа>);

T={ "int","main","double","float","for", '+', '-', '*', '/', ';','=', ',', '>', '<', '(', ')', '}', '{', "++", "--", "<=", ">=","==","!=",id,lit,expr}

N={<программа>, <список_операторов>, <оператор>,<объявление>, <тип>, <список_перемменных>, <присваивание>, <операнд>, <знак>, <объявление&присваивание>, <список_присваивания>, <цикл>, <лог>, <блок_операторов>}

P={<программа>::=int main(){<список_операторов>}

<список_операторов>::=<оператор> | <оператор><список_операторов>

<оператор>::=<объявление>|<присваивание>;|<объявление&присваивание> | <цикл>

<объявление>::=<тип> <список_переменных>;

<тип>::=int|double|float

<список_переменных>::=id | <список_перемменных>, id

<присваивание>::=id=<операнд>|id=<операнд><знак><операнд>|id=<операнд>++| id=<операнд>--|id=<expr>

<операнд>::=id|lit

<знак>::=+|-|*|/

<объявление&присваивание>::=<тип><список_присваивания>;

<список_присваивания>::=<присваивание>|<список_присваивания>,<присваивание>

<цикл>::=for(<инициализация><условие><итерация>)<блок_операторов>

<инициализация>::=<присваивание>;| <объявление&присваивание>

<условие>::= id<лог><операнд>;

<итерация>::= id++|id--| <присваивание>| *E*

<лог>::= >|<|<=|>=|==|!=

<блок_операторов>::=<оператор>|{<список_операторов>}

Для проверки соответствия полученной грамматики требуемым правилам, можно сформировать последовательный процесс вывода, который симулирует работу снизу вверх. Проверка начинается с главного символа грамматики. Далее необходимо последовательно заменять нетерминалы в полученной цепочке, пока не получится корректный фрагмент кода, как представленный в примере.

Пример формирования цепочки вывода:

<программа>::= int main(){<список_операторов>}=> int main(){<оператор>;<список_операторов>}=> int main(){<объявление><список_операторов>}=>int main(){<тип><список_переменных>;<список_операторов>}=> int main(){int <список_переменных>,id;<список_операторов>}=> int main(){<тип> id,id;<список_операторов>}=> int main(){int id,id;<список_операторов>}=> int main(){int id,id; <присваивание>; <список_операторов>}=> int main(){int id,id; id=<операнд>; <список_операторов>}=> int main(){ int id,id; id=lit; <список_операторов>}=> int main(){ int id,id; id=lit; <цикл>}=> int main(){ int id,id; id=lit; for(<присваивание>; id<лог><операнд>; id++;)<блок_операторов>}=> int main(){ int id,id; id=lit; for(id=<операнд>; id<lit; id++;){<список_операторов>}}=> int main(){ int id,id; id=lit; for(id=lit; id<lit; id++;){<опер>;<список_операторов>}}=> int main(){ int id,id; id=lit; for(id=lit; id<lit; id++;){<оператор>;<оператор>}}=> int main(){ int id,id; id=lit; for(id=lit; id<lit; id++;){<присваивание>;<присваивание>;}}=> int main(){ int id,id; id=lit; for(id=lit; id<lit; id++;){id=<операнд>;id=<expr>;}}=> int main(){ int id,id; id=lit; for(id=lit; id<lit; id++;){id=lit;id=<expr>;}}

Далее необходимо сравнить вывод с исходным кодом.

Таблица 1 – Сравнение полученной цепочки и анализируемого фрагмента

Анализируемый фрагмент	Полученная цепочка вывода

int main ()
{ int a,b;
c=0;
for (b=1; b<5; b++) {
b = 5;
c=c+b*3; }
}
	int main()
	{ int id,id;
	id=lit;
	for(id=lit; id<lit; id++;){
	id=lit;
	id=<expr>;}
	}

Условием окончания цепочки вывода является получение сентенции, то есть строки, содержащей только символы множества терминалов.

2.4 Разработка восходящего анализатора

После успешного тестирования грамматики, следующим этапом является создание восходящего анализатора класса LR(k) грамматик.

Для этого необходимо:

1. Создать граф состояний;
2. Построить таблицу решений для LR-анализатора.

Для создания графа состояний используется следующий процесс:

1. В начальное состояние вносятся все правила, которые можно вывести из генерирующего символа грамматики, и перед самым левым символом в правой части ставится маркер;
2. Для каждого нетерминала, который помечен маркером, вносятся правила, которые он может генерировать, с маркером перед самым левым символом в правой части;

3. Список правил, полученных выполнением п. 1-2, называется состоянием;

4. Если маркер в правиле состояния находится после последнего символа, то в столбец "переход" записывается "нет перехода", а состояние отмечается как конечное.

5. Если в строке, соответствующей состоянию D, ячейка "правила" заполнена, а ячейка "переход" пуста, то в нее записывается номер следующего свободного состояния n. Если в этом состоянии есть другие строки с таким же маркером, то в ячейки "переход" этих строк также записывается номер n.

6. Все строки из текущего состояния D, у которых в столбце "переход" записан n и маркер в них, переносятся на один символ вправо и записываются в следующее свободное состояние n.

7. Повторяются действия 2-3, пока все ячейки столбца "Переход" не будут заполнены.

Посредством выполнения этого алгоритма был построен граф состояний:

Таблица 2 – Граф состояний

Состояние	Пред. Состояние	Правила	Переход
0	-	<программа>::=●int main(){<список_операторов>}	1
1	0	<программа>::=int ●main(){<список_операторов>}	2
2	1	<программа>::=int main● (){<список_операторов>}	3
3	2	<программа>::=int main (●){<список_операторо>}	4
4	3	<программа>::=int main ()●{<список_операторов>}	5
5	4	<программа>::=int main(){●<список_операторов>} <список_операторов>::=●<оператор> <список_операторов>::=●<оператор><список опеаторов> <оператор>::=●<объявление> <оператор>::= ●<присваивание>;	6 7 7 8

		<оператор>::= ●<объявление&присваивание> <оператор>::= ●<цикл> <объявление>::=●<тип> <список_переменных>; <тип>::= ●int <тип>::=●double <тип>::=●float <присваивание>::=●id=<операнд> <присваивание>::=●id=<операнд><знак><операнд> <присваивание>::=●id=<операнд>++ <присваивание>::=●id=<операнд>-- <присваивание>::=●id=<expr> <объявление&присваивание>::=●<тип><список_пр сваивания>; <цикл>::=●for(<инициализация><условие><итерац я>)<блок_операторов>	9 10 11 12 13 14 15 16 16 16 16 16 12 17
6	5	<программа>::=int main(){<список_операторов>●}	18
18	6	<программа>::=int main(){<список_операторов>}●	X
7	5,53	<список_операторов>::=<оператор>● <список_операторов>::=<оператор>●<список_опер торов> <список_операторов>::=●<оператор> <список_операторов>::=●<оператор><список операторов> <оператор>::=●<объявление> <оператор>::= ●<присваивание>; <оператор>::= ●<объявление&присваивание> <оператор>::= ●<цикл> <объявление>::=●<тип> <список_переменных>; <тип>::= ●int <тип>::=●double <тип>::=●float <присваивание>::=●id=<операнд> <присваивание>::=●id=<операнд><знак><операнд> <присваивание>::=●id=<операнд>++ <присваивание>::=●id=<операнд>-- <присваивание>::=●id=<expr>	X 59 60 60 8 9 10 11 12 13 14 15 16 16 16 16 16

		<объявление&присваивание>::=●<тип><список присв>; <цикл>::=●for(<инициализация><условие><итерация>)<блок_операторов>	12 17
8	5	<оператор>::=<объявление>●	X
9	5,23	<оператор>::= <присваивание>●;	19
19	9	<оператор>::= <присваивание>;●	X
10	5,23	<оператор>::= <объявление&присваивание>●	X
11	5	<оператор>::= <цикл>●	X
12	5	<объявление>::=<тип>●<список_переменных>; <список_переменных>::=●id <список_переменных>::= ●<список_переменных>id	20 21 20
13	5	<тип>::= int●	X
14	5	<тип>::= double●	X
15	5	<тип>::= float●	X
16	5,33	<присваивание>::=id●=<операнд> <присваивание>::=id●=<операнд><знак><операнд> <присваивание>::=id●=<операнд>++ <присваивание>::=id●=<операнд>-- <присваивание>::=id●=<expr>	22 22 22 22 22
17	5	<цикл>::=for●(<инициализация><условие><итерация>)<блок опер>	23
20	12	<объявление>::=<тип><список_переменных>●; <список_переменных>::= <список_переменных>●id	24 25
21	12	<список_переменных>::=id●	X
22	16	<присваивание>::=id=●<операнд> <присваивание>::=id=●<операнд><знак><операнд> <присваивание>::=id=●<операнд>++ <присваивание>::=id=●<операнд>-- <присваивание>::=id=●<expr> <операнд>::=●id <операнд>::=●lit	26 26 26 26 27 31 32
23	17	<цикл>::=for(●<инициализация><условие><итерация>)<блок_операторов>	28

		<инициализация>::=●<присваивание>; <инициализация>::=●<объявление&присваивание>	9 10
24	20	<объявление>::=<тип><список_переменных>;●	X
25	20	<список_перемеменных>::= <список_переменных>,● id	29
26	22,45	<присваивание>::=id=<операнд>● <присваивание>::=id=<операнд>●<знак><операнд> <присваивание>::=id=<операнд>●++ <присваивание>::=id=<операнд>●-- <знак>::=●+ <знак>::=●- <знак>::=●* <знак>::=●/	X 35 36 37 38 39 40 41
27	22	<присваивание>::=id=<expr>●	X
28	23	<цикл>::=for(<инициализация>●<условие><итерация>)<блок_операторов> <условие>::= ●id<лог><операнд>;	33 34
29	25	<список_переменных>::= <список_переменных>, id●	X
31	26,35,45	<операнд>::=id●	X
32	26,35,45	<операнд>::=lit●	X
33	28	<цикл>::=for(<инициализция><условие>●<итерация>)<блок_операторов> <итерация>::= ●id++ <итерация>::=●id-- <итерация>::=●<присваивание> <присваивание>::=●id=<операнд> <присваивание>::=●id=<операнд><знак><операнд> <присваивание>::=●id=<операнд>++ <присваивание>::=●id=<операнд>-- <присваивание>::=●id=<expr> <итерация>::=●	42 43 43 44 43 43 43 43 43 X
34	28	<условие>::= id●<лог><операнд>; <лог>::= ●>	45 46

		<лог>::= ●< <лог>::= ●<= <лог>::= ●>= <лог>::= ●== <лог>::= ●!+	47 48 49 50 51
35	26	<присваивание>::=id=<операнд><знак>●<операнд> <операнд>::=●id <операнд>::=●lit	52 31 32
36	26	<присваивание>::=id=<операнд>++●	X
37	26	<присваивание>::=id=<операнд>--●	X
38	26	<знак>::=+●	X
39	26	<знак>::=-●	X
40	26	<знак>::=*●	X
41	26	<знак>::=/●	X
42	33	<цикл>::=for(<инициализация><условие><итерация>●)<блок_операторов>	53
43	33	<итерация>::= id●++ <итерация>::=id●-- <присваивание>::=id●=<операнд> <присваивание>::=id●=<операнд><знак><операнд> <присваивание>::=id●=<операнд>++ <присваивание>::=id●=<операнд>-- <присваивание>::=id●=<expr>	57 58 22 22 22 22 22
44	33	<итерация>::=<присваивание>●	X
45	34	<условие>::= id<лог>●<операнд>; <операнд>::=●id <операнд>::=●lit	54 31 32
46	34	<лог>::= >●	X
47	34	<лог>::= <●	X
48	34	<лог>::= <=●	X
49	34	<лог>::= >=●	X
50	34	<лог>::= ==●	X
51	34	<лог>::= !+●	X
52	35	<присваивание>::=id=<операнд><знак><операнд>●	X
53	42	<цикл>::=for(<инициализация><условие><итерация>)●<блок_операторов>	55

		<блок_операторов>::=●<опер> <блок_операторов>::=●{<список_операторов>}	7 5
54	45	<условие>::= id<лог><операнд>●;	56
55	53	<цикл>::=for(<инициализация><условие><итерация>)<блок_операторов>●	X
56	54	<условие>::= id<лог><операнд>;●	X
57	43	<итерация>::= id++●	X
58	43	<итерация>::=id--●	X
59	7	<список_операторов>::=<оператор><список_операторов>●	X
60	7	<список_операторов>::=<оператор>● <список_операторов>::=<операторов>●<список_операторов>	X 7

Следующим шагом в разработке восходящего анализатора является создание решающей таблицы. Это делается на основе анализа строк, связанных с состоянием с тем же именем. Чтобы создать таблицу принятия решений, нужно следовать следующему алгоритму:

1. В колонке "Стек разбора" для начального состояния должен быть указан порождающий символ грамматики с действием "КОНЕЦ", а также строка "ε" с действием "СДВИГ".

2. Все строки, связанные с анализируемым состоянием, просматриваются. В столбец "Стек разбора" записывается элемент, находящийся перед маркером, а в столбец "действие" записывается базовое действие - "СДВИГ" или "СВЕРТКА". Если все строки являются переходами, то базовым действием будет "СДВИГ", а если все строки помечены как конечные, то базовым действием будет "СВЕРТКА".

3. Если в состоянии присутствуют и строки с переходами, и строки с признаком "нет перехода", то в этом состоянии должны быть указаны оба действия - "СДВИГ" и "СВЕРТКА".

На основании вышеприведенного алгоритма была построена решающая таблица:

Таблица 3 - Решающая таблица восходящего анализатора

Состо-яние	Стек	Вход	Действие
0	<программа> E int		Конец Сдвиг →1
1	int main		Сдвиг →2
2	main (		Сдвиг →3
3	( )		Сдвиг →4
4	) {		Сдвиг →5
5	{ <список_операторов> <оператор> <объявление> <присваивание> <объявление&присваивание> <цикл> <тип> int double float id for		Сдвиг →6 →7 →8 →9 →10 →11 →12 →13 →14 →15 →16 →17
6	<список_операторов> }		Сдвиг →18
7	<оператор> <оператор> <список_операторов> <объявление> <присваивание> <объявление&присваивание> <цикл> <тип>	} Int,double,float,id, for	Свертка(-1,<список_операторов> Сдвиг →5 →8 →9 →10 →11 →12

	int double float id for		→13 →14 →15 →16 →17
8	<объявление>		Свертка(-1,<оператор>)
9	<присваивание> ;		Сдвиг →19
10	<объявление&присваивание>		Свертка(-1,<оператор>)
11	<цикл>		Свертка(-1,<оператор>)
12	<тип> <список_переменных> id		Сдвиг →20 →21
13	int		Свертка(-1,<тип>)
14	double		Свертка(-1,<тип>)
15	float		Свертка(-1,<тип>)
16	id =		Сдвиг →22
17	for (		Сдвиг →23
18	}		Свертка(-7,<программа>)
19	;		Свертка(-2,<оператор>)
20	<список_переменных> ; ,		Сдвиг →24 →25
21	id		Свертка(-1, <список_переменных>)
22	= <операнд> <expr> id lit		Сдвиг →26 →27 →31 →32
23	<инициализация> <присваивание> <объявление&присваивание>		→28 →9 →10
24	;		Свертка(-3,<объявление>)
25	, id		Сдвиг →29
26	<операнд> <операнд> <знак>	; ++,--,+,-,*,/	Свертка(-3,<присваивание>) Сдвиг →35

	++ -- + - * /		→36 →37 →38 →39 →40 →41
27	<expr>		Свертка(-3,<присваивание>)
28	<инициализация> <условие> id		Сдвиг →33 →34
29	id		Свертка(-3,<список_переменных>)
31	id		Свертка(-1,<операнд>)
32	lit		Свертка(-1,<операнд>)
33	<условие> <условие> <итерация> id <присваивание>	id)	Сдвиг Свертка(0,<итерация>) →42 →43 →44
34	id <лог> > < <= >= == !=		Сдвиг →45 →46 →47 →48 →49 →50 →51
35	<знак> <операнд> id lit		Сдвиг →52 →31 →32
36	++		Свертка(-4,<присваивание>)
37	--		Свертка(-4,<присваивание>)
38	+		Свертка(-1,<знак>)
39	-		Свертка(-1,<знак>)
40	*		Свертка(-1,<знак>)
41	/		Свертка(-1,<знак>)
42	<итерация>)		Сдвиг →53
43	++ -- =		→57 →58 →53

44	<присваивание>		Свертка(-1,<итерация>)
45	<лог> <операнд> id lit		Сдвиг →54 →31 →32
46	>		Свертка(-1,<лог>)
47	<		Свертка(-1,<лог>)
48	<=		Свертка(-1,<лог>)
49	>=		Свертка(-1,<лог>)
50	==		Свертка(-1,<лог>)
51	!=		Свертка(-1,<лог>)
52	<операнд>		Свертка(-5,<присваивание>)
53	) <блок_операторов> <оператор> {<список_операторов>}		Сдвиг →55 →7 →5
54	<операнд> ;		Сдвиг →56
55	<блок_операторов>		Свертка(-7,<цикл>)
56	;		Свертка(-4,<условие>)
57	++		Свертка(-2, <итерация>)
58	--		Свертка(-2, <итерация>)
59	<список_операторов>		Свертка(-2, <список_операторов>)
60	<оператор> <оператор> <список_операторов>	} Int,double,float,id, for	Свертка(-1,<список_операторов>) Сдвиг →7

При построении решающей таблицы было выявлено, что в некоторых состояниях необходимо просматривать следующий символ входного потока для принятия решения. Таким образом k=1, а грамматика принадлежит к классу LR(1).

3. Разработка архитектуры системы и алгоритмов

3.1 Алгоритмы лексического анализа

Лексический анализ в программе разделен на две части. Первым этапом является выделение всех лексем заданного фрагмента. Процесс выделения всех лексем выглядит следующим образом:

Заданный фрагмент передается в подпрограмму в качестве строки, соответственно все escape-символы, такие как перенос на новую строку в подпрограмму не передаются. Далее с помощью специального метода определенного класса выполняется разбор входной строки на отдельные фрагменты, шаблоны которых заранее прописываются в программе. На выходе получаем сформированный список выделенных лексем. При разборе входного потока алгоритм пропускает пробелы, табуляцию и перенос строки.

Вторым этапом является классификация лексем. Она основывается на списке лексем из предыдущего этапа, а ее задачей является разделение лексем на ключевые слова, разделители, идентификаторы и литералы. Классификация лексем происходит по следующему принципу: на вход следующей подпрограмме передается список всех лексем. Далее, с помощью специального метода, входной список группируется на 4 набора, в зависимости от типа:

– Литералы – подпрограмма проверяет первый символ литерала, если это число, то заносит его в набор с ключом “Литерал”. Стоит отметить, что в набор попадают все литералы, прошедшие проверку, включая и дубликаты, но в последующих процедурах при добавлении в результирующую таблицу дубликаты удаляются. Данный процесс идентичен для всех последующих наборов лексем;

– Ключевые слова – проверяется, соответствует ли данное ключевое слово в шаблоне, содержащем только ключевые слова. Если да, то данная лексема отправляется в набор с ключом “Ключевое слово”;

– Идентификаторы – подпрограмма проверяет лексемы, и если лексема, первый символ которой начинается с буквы или символа ‘_’, найдена, то она добавляется в набор с ключом “Идентификатор”;

– Разделители включают в себя все оставшиеся лексемы и заносятся в набор с ключом “Разделители”.

3.2 Программирование системы

3.2.1 Программирование архитектуры системы

В разрабатываемом приложении был использован язык C# и среда разработки Visual Studio 2022. Графический интерфейс приложения будет реализован с помощью фреймворка WinForms. Поскольку приложение будет состоять из одного окна, то для организации информации будет использован компонент TabControl. Программа способна анализировать фрагменты кода, введенные пользователем через компонент TextBox. Для управления функциями программы будут использованы компоненты Button и их Click-события. Результаты работы программы будут отображены в виде таблицы на форме с помощью компонента DataGridView.

4 Руководство программиста

Решение включает в себя следующие классы и компоненты:

1. Program.cs – класс, в котором содержится метод Main(), с которого и начинается жизненный цикл программы.
2. Variables.cs – класс-шаблон для заполнения поддерживаемых лексем для ключевых слов, одиночных и двойных разделителей.

Данный класс включает в себя следующие поля и конструктор:

public List<string> SingleSeparators, DoubleSeparators, KeyWords – поля, хранящие в себе шаблоны для выше сказанных лексем.

public Variables() – конструктор, служащий для заполнения выше перечисленных полей.

3. Helper.cs – статический класс, в котором реализуется обработка информации.

Данный класс включает следующие методы:

public static List<string> FirstProcess(string inputString, Variables variables) – данный метод выполняет разделение строки "inputString" на отдельные Matches, то есть совпадения, используя регулярные выражения. Он принимает в качестве аргументов строку "inputString" и объект "variables". Метод, используя класс Regex, сопоставляет строку "inputString" с регулярным выражением, которое учитывает разделители и идентификаторы. В качестве результата метод возвращает список выделенных лексем.

public static string GetStringType(string t, Variables variables) – данный метод принимает два параметра: строковый тип t и объект variables. Он использует свойства первого символа в строке t и сверяет его с ключевыми словами в объекте variables для определения типа строки. Если первый символ является цифрой, метод возвращает строку "Литерал". Если строка t содержится в списке ключевых слов variables.KeyWords, метод возвращает

"Ключевое слово". Если первый символ является буквой или знаком подчеркивания, метод возвращает "Идентификатор". В противном случае метод возвращает "Разделитель".

public static IEnumerable<IGrouping<string, string>> SecondProcess(string inputString, Variables variables) – данный метод принимает два аргумента: строку inputString и объект variables. Метод использует метод FirstProcess, чтобы разделить строку inputString на лексемы и вернуть список этих лексем. Затем метод SecondProcess использует метод GetStringType для получения типа каждой лексемы. Наконец, метод возвращает группу лексем, сгруппированных по их типам. Возвращаемый тип коллекцию коллекций лексем, сгруппированных по типу.

4. MainForm.cs – код класса формы. В данном классе идет обработка событий, посылаемых при нажатии на соответствующие кнопки из формы, таких как формирование общего списка, или начало лексического анализа.

Исходный код программы можно посмотреть в Приложении А.

5 Тестирование

После реализации программного продукта немаловажным шагом является его тестирование. Программа должна корректно реагировать на производимые пользователем действия.

Первым этапом тестирования является выделения ввод фрагмента кода в «Редактор кода». Если два варианта заполнения редактора кода: ввод вручную или открытие файла с расширением .txt. Ввод фрагмента кода представлен на Рисунке 1, а ввод кода через открытие файла представлен на Рисунке 2.

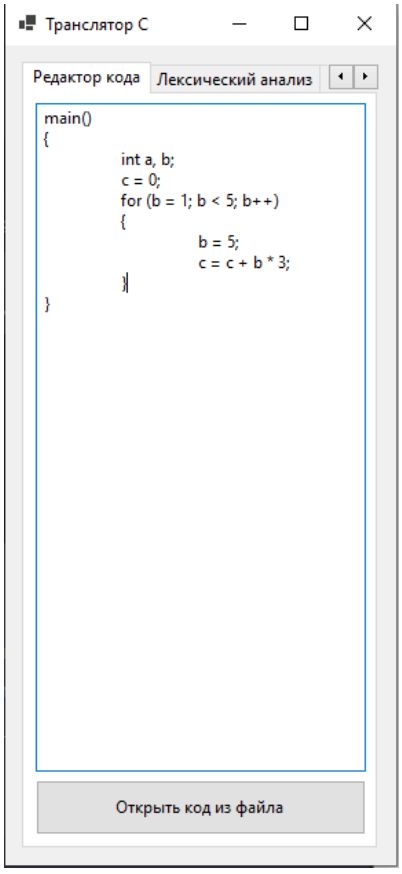

Рисунок 1 – Введенный фрагмент кода в «Редактор кода»

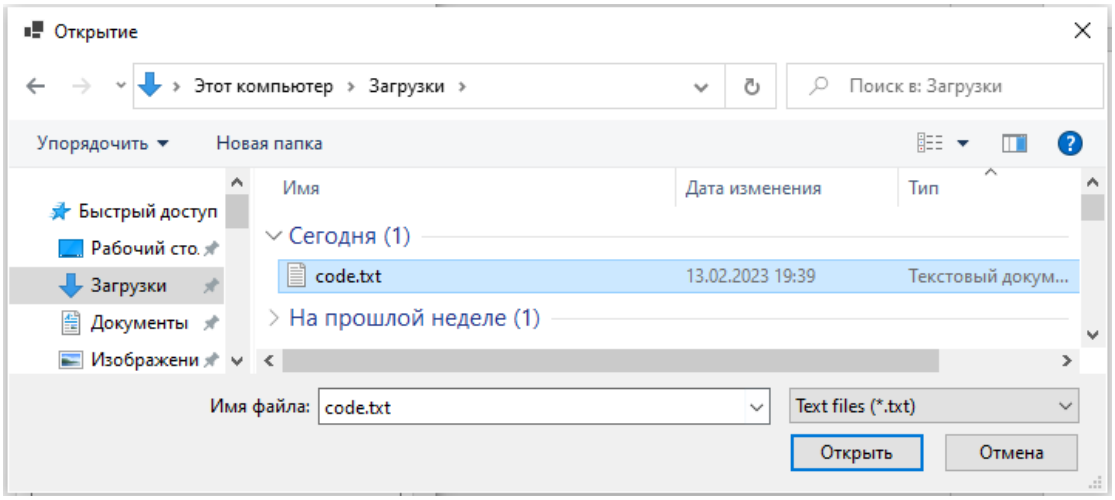

Рисунок 2 – Заполнение редактора кода через открытие файла

Следующим этапом тестирования является составление лексического анализа из заданного фрагмента кода. После того как пользователь ввел фрагмент в «Редактор кода», с помощью компонента TabControl нужно перейти на вкладку «Лексический анализ». После нажатия на кнопку «Начать лексический анализ» заполнится компонент DataGridView всеми лексемами, которые были найдены. Данный процесс показан на Рисунке 3. Если при нажатии на эту кнопку редактор кода не был заполнен, на экран покажется компонент MessageBox с сообщением о том, что редактор был пуст.

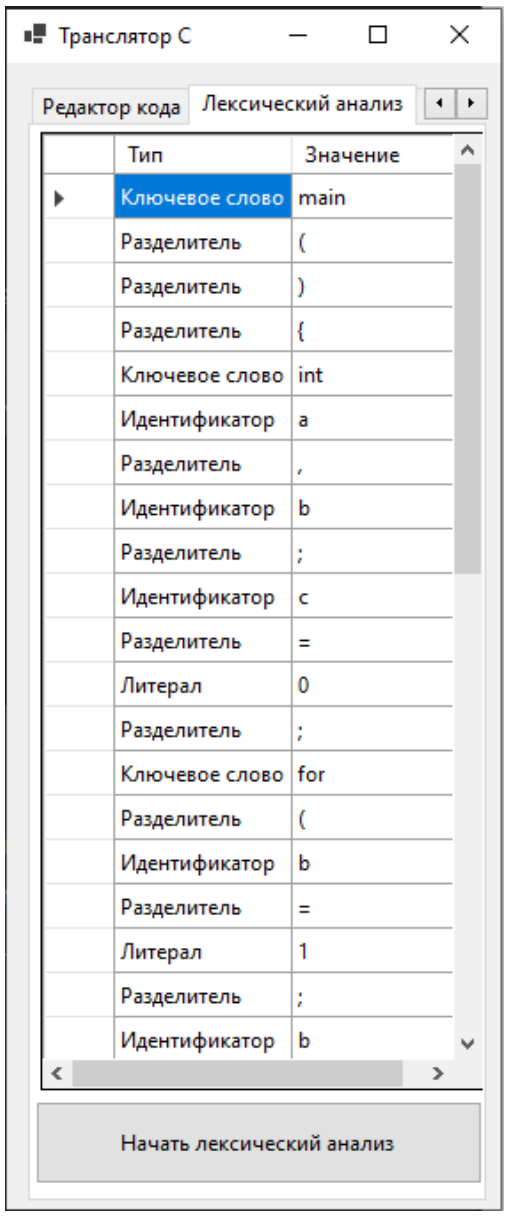

Рисунок 3 – Демонстрация лексического анализа

Наконец, завершающим этапом является классификация лексем, в результате которой все лексемы их лексического анализа будут распределены по своим таблицам, исключая дубликаты, и будет сгенерирована общая таблица, где в последовательном порядке будут выведены лексемы в виде «Таблица лексемы – индекс лексемы в данной таблице». Результат работы показан на Рисунке 4.

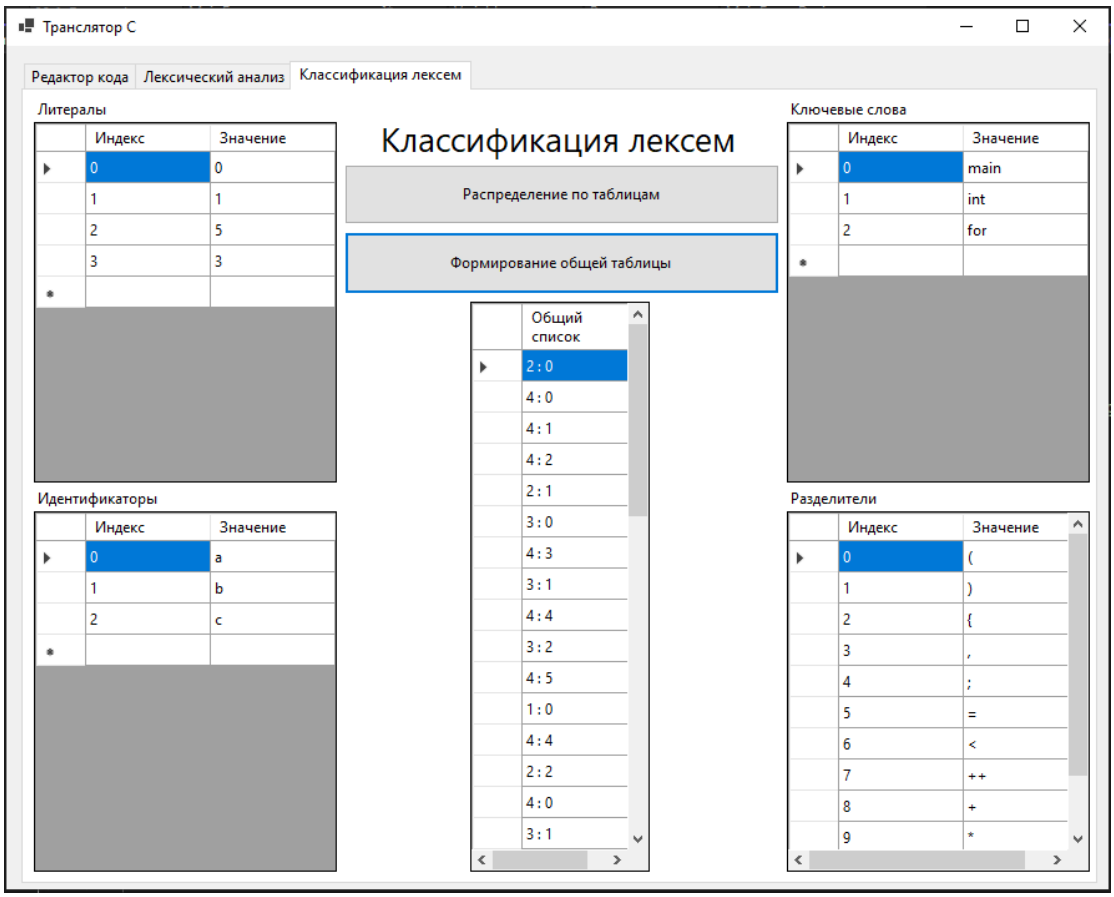

Рисунок 4 – Результат классификации лексем

Заключение

В ходе выполнения курсовой работы был произведен анализ технического задания, сформулированы требования к разрабатываемой программе. Была созданы алгоритмы нахождения лексем и их классификации. Так же была разработана грамматика языка и восходящий анализатор LR(1) в виде графа состояний и результирующей таблице. Восходящий анализатор и разбор сложных арифметических выражений не были реализованы в программе.

Реализованное приложение отвечает основным требованиям, предъявленным в задании к курсовой работе. Программа была разработана с использованием эффективных алгоритмов и механизмов отлова исключений. Приложение было реализовано в среде Visual Studio Community Edition 2022.

В дальнейшем программа может быть улучшена, например, возможность программной реализации восходящего анализатора и разбор сложных арифметических выражений.

Список использованной литературы

1. Шульга, Т. Э. Теория автоматов и формальных языков : учебное пособие / Т. Э. Шульга. — Саратов : Саратовский государственный технический университет имени Ю.А. Гагарина, ЭБС АСВ, 2015. — 104 с.

2. Алымова, Е. В. Конечные автоматы и формальные языки : учебник / Е. В. Алымова, В. М. Деундяк, А. М. Пеленицын. — Ростов-на-Дону, Таганрог : Издательство Южного федерального университета, 2018. — 292 с.

3. Малявко, А. А. Формальные языки и компиляторы : учебник / А. А. Малявко. — Новосибирск : Новосибирский государственный технический университет, 2014. — 431 с.

Приложение А. Ссылка на исходный код программы

Исходный код приложение можно просмотреть и скачать, при помощи системы контроля версий GitHub. Ссылка на репозиторий: https://github.com/Muntissa/SimpleCompilatorClang