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 С. 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 6](#_Toc124169833)

[1. Анализ технического задания 7](#_Toc124169834)

[2. Описание грамматики языка 8](#_Toc124169835)

[2.1 Описание языка 8](#_Toc124169836)

[2.2 Описание синтаксиса операторов и основных конструкций 8](#_Toc124169837)

[2.3 Разработка грамматики языка 9](#_Toc124169838)

[2.4 Разработка восходящего анализатора 12](#_Toc124169839)

[3. Разработка архитектуры системы и алгоритмов 23](#_Toc124169840)

[3.1 Алгоритмы лексического анализа 23](#_Toc124169841)

[3.2 Программирование системы 24](#_Toc124169842)

[3.2.1 Программирование архитектуры системы 24](#_Toc124169843)

[4 Руководство программиста 26](#_Toc124169844)

[5 Тестирование 28](#_Toc124169845)

[Заключение 31](#_Toc124169846)

[Список использованной литературы 32](#_Toc124169847)

[Приложение А. Ссылка на исходный код программы 33](#_Toc124169848)

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 C, 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

C - 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 Т, 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 Р, 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>);

Т={ "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\_перемеменных>, 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>;  <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\_присваивания>;  <цикл>::=●for(<инициализация><условие><итерация>)<блок\_операторов> | 6  7  7  8  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\_операторов>  <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>  <объявление&присваивание>::=●<тип><список присв>;  <цикл>::=●for(<инициализация><условие><итерация>)<блок\_операторов> | X  59  60  60  8  9  10  11  12  13  14  15  16  16  16  16  16  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>  <инициализация>::=●<присваивание>;  <инициализация>::=●<объявление&присваивание> | 28  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>::=●/ | Х  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>::= ●>  <лог>::= ●<  <лог>::= ●<=  <лог>::= ●>=  <лог>::= ●==  <лог>::= ●!+ | 45  46  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>  <блок\_операторов>::=●<опер>  <блок\_операторов>::=●{<список\_операторов>} | 55  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++● | Х |
| 58 | 43 | <итерация>::=id--● | Х |
| 59 | 7 | <список\_операторов>::=<оператор><список\_операторов>● | Х |
| 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 РАЗБОРА", 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

|  |  |  |  |
| --- | --- | --- | --- |
| Состояние | 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 | }  Int,double,float,id, for | Свертка(-1,<список\_операторов>)  Сдвиг  →5  →8  →9  →10  →11  →12  →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. Алгоритм выделения лексем следующий:

1. Фрагмент проверяется на наличие в нем только английских букв;
2. Далее каждый символ проверяется на соответствие букве, цифре или разделителю и происходит переход в одно из состояний. Если была обнаружена буква или цифра, то происходит дальнейшее считывание входного потока до столкновения с символом, не являющимся буквой или цифрой соответственно. В случае же обнаружения разделителя считывается только следующий символ. В обоих случаях, при считывании более одного символа, маркер во входном потоке сдвигается на количество обнаруженных символов.
3. На каждой итерации, обнаруженная лексема заносится в список.
4. Пункты с 2 по 3 повторяются до конца входного потока.

При разборе входного потока алгоритм пропускает пробелы, табуляцию и перенос строки. В случае, если исходный код содержал неанглийские буквы или при разборе встретился неизвестный символ, то выполнение программы прерывается.

Вторым этапом является классификация лексем. Она основывается на списке лексем из предыдущего этапа, а ее задачей является разделение лексем на служебные слова, разделители, идентификаторы и литералы. Классификация лексем происходит по следующему принципу: на вход получается список лексем; далее у каждой лексемы берется тип и в зависимости от него осуществляется переход в одно из состояний:

* Идентификатор - проверяется соответствует ли лексема одному из служебных слов(int,main,float,double,for,if) и в случае соответствия лексема заносится в результирующую таблицу с типом служебного слова и ее индексом. В случае если лексема не соответствует ни одному из служебных слов, то происходит проверка является ли лексема уже занесенной переменной или нет. Если переменная уже известна, то в результирующую таблицу заносится ее индекс и тип переменная, а в случае если такой переменной еще нет, она заносится в таблицу переменных и в результирующую таблицу заносится ее индекс и тип.
* Литерал – проверяется известен ли уже такой литерал и в случае, если известен в результирующую таблицу добавляется его индекс и тип. Если такой литерал не известен, то он заносится в таблицу литералов и в результирующую таблицу заносится его индекс и тип.
* Разделитель – проверяется соответствует ли разделитель одиночному или двойному разделителю и в соответствии с результатом в результирующую таблицу заносится его индекс и тип.

3.2 Программирование системы

3.2.1 Программирование архитектуры системы

Для реализации разрабатываемого приложения был выбран язык C# и среда разработки Visual Studio 2022. Программа будет предоставлять графический интерфейс на основе фреймворка WinForms. Так как приложение будет состоять из одного окна, то для упорядочивания информации будет использован визуальный компонент tabControl. Программа анализирует введенный пользователем фрагмент кода, для того что бы пользователь мог вводить его с клавиатуры был добавлен компонент textBox. Для управления функциями программы были добавлены кнопки button. Результаты работы программы выводятся на форму в виде таблицы, при помощи компонента dataGridView. При возникновении ошибок, информация о них выводится в компонент MessageBox.

4 Руководство программиста

Алгоритм выделения лексем из кода реализован в классе LexemFinder. Данный класс содержит поле public static char[] razd , которое содержит одиночные разделители и поле public static string[] dividers , содержащее двойные разделители. Поле Regex ASCIILettersOnly необходимо для проверки фрагмента на английские символы. Для хранения промежуточных данных используется поле string buffer , а для хранения всего фрагмента используется поле string buffer. Класс содержит конструктор со следующей сигнатурой public LexemFinder(String source) и необходим для получения анализируемого фрагмента. Класс имеет всего один метод public List<WordType> find() , который не принимает входных параметров и выполняет выделение лексем из фрагмента, заданного в конструкторе класса. Данный метод возвращает список выделенных лексем.

Для хранения списка лексем используется List классов WordType. Данный класс позволяет хранить связку лексемы и ее типа. Лексема хранится в поле string Word , а ее тип хранится в поле string Type. Конструктор класса имеет сигнатуру public WordType(string word, string type) и предназначен для заполнения полей. В классе реализованы методы public string getWord() и public string getType() для получения лексемы и ее типа соответственно.

Алгоритм классификации лексем релизован в классе ClassifyLexem. Данный класс также содержит поля для одиночных и двойных разделителей char[] oneRazd и string[] dubRazd, а так же поле для выделенных лексем List<WordType> lexems , поле для заданных служебных слов public static string[] keyWords, поле для хранения классифицированных переменных public static List<string> variables и поле для классифицированных литералов public static List<int> literals, а так же поле для хранения результирующей таблицы List<WordType> result. В конструкторе класса с сигнатурой public ClassifyLexem(char[] odinrazd, string[] dubrazd, List<WordType> impLexems ) задаются списки одиночных и двойных разделителей, а так же список выделенных лексем. В классе создан метод public List<WordType> classificate(), который выполняет классификацию лексем и не принимает входных параметров, работая с заданными полями класса. Данный метод возвращает список классифицированных лексем.

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 дейтсвия.

Первым этапом тестирования является выделения лексем из заданного фрагмента. Если пользователь ввел текст, состоящий только из английских букв, то программа выделит все лексемы и выведет их на dataGridView во вкладке «Лекс. Анализ»(Рисунок 1). В случае же, если данное условие не выполнится, то пользователь получит сообщение об ошибке(Рисунок 2).

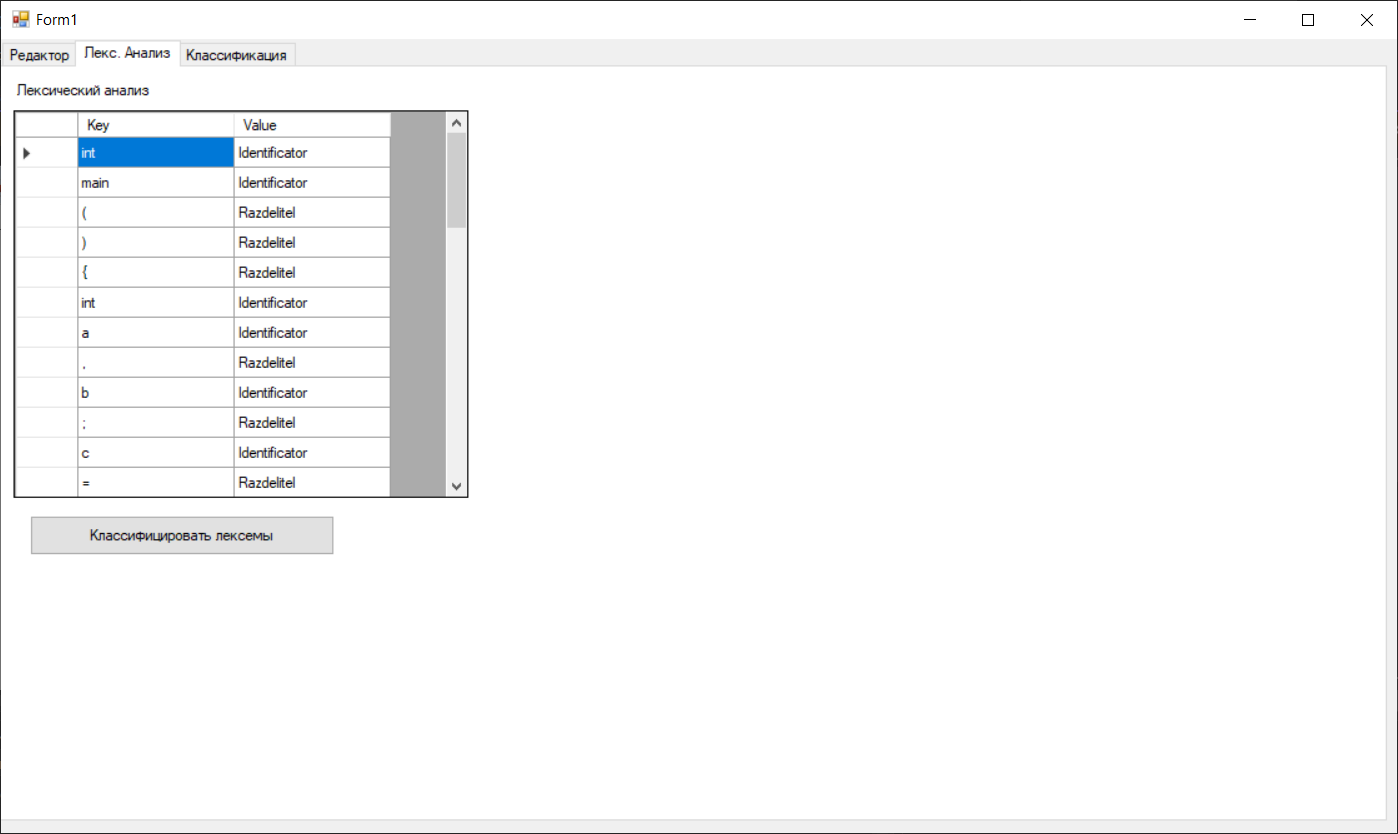


Рисунок 1 – Успешно выведенные лексемы

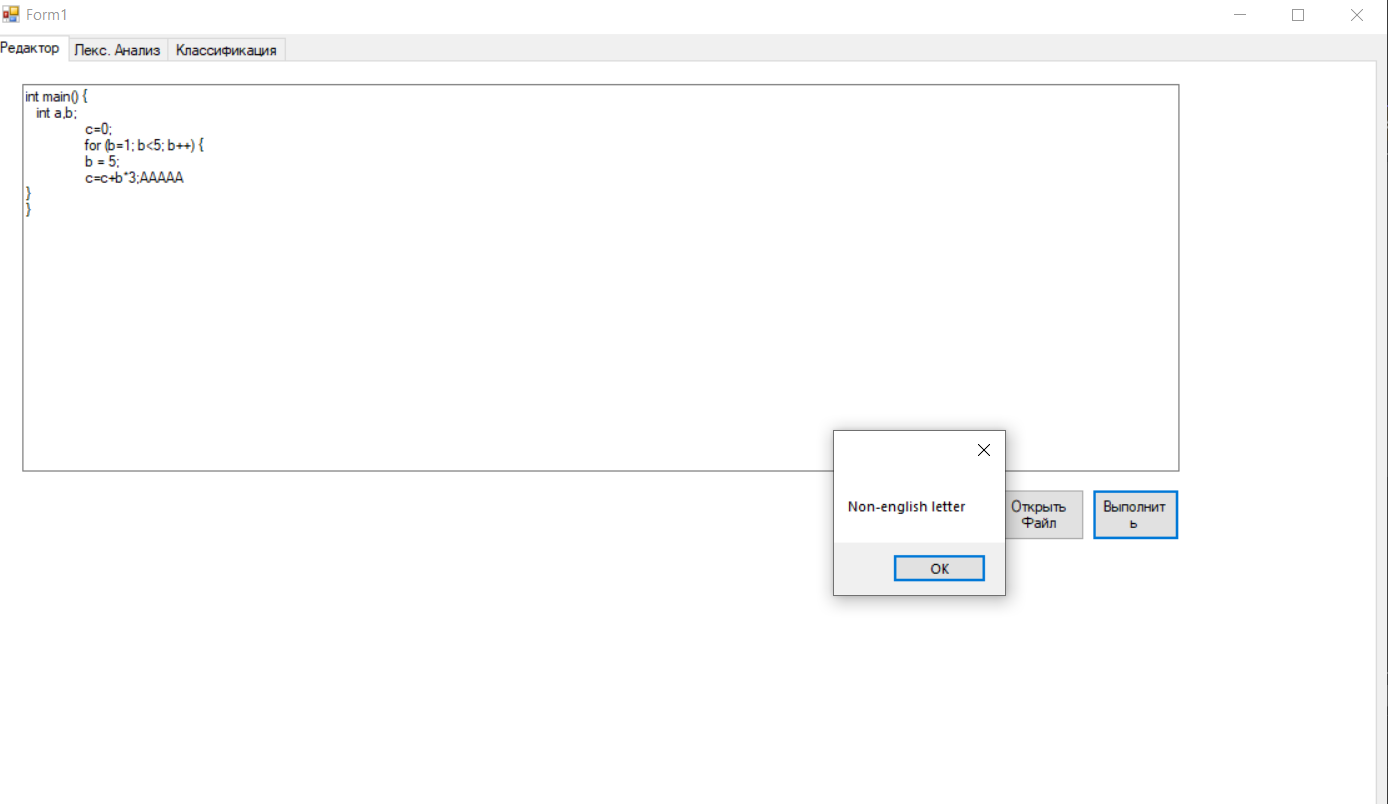


Рисунок 2 – Сообщение о некорректном вводе

Следующим этапом является тестирование классификации лексем. При успешном тестировании произойдет переход на вкладку, где представлены таблицы служебных слов, одиночных и двойных разделителей, литералов и переменных, а также таблица классифицированных лексем(Рисунок 3).

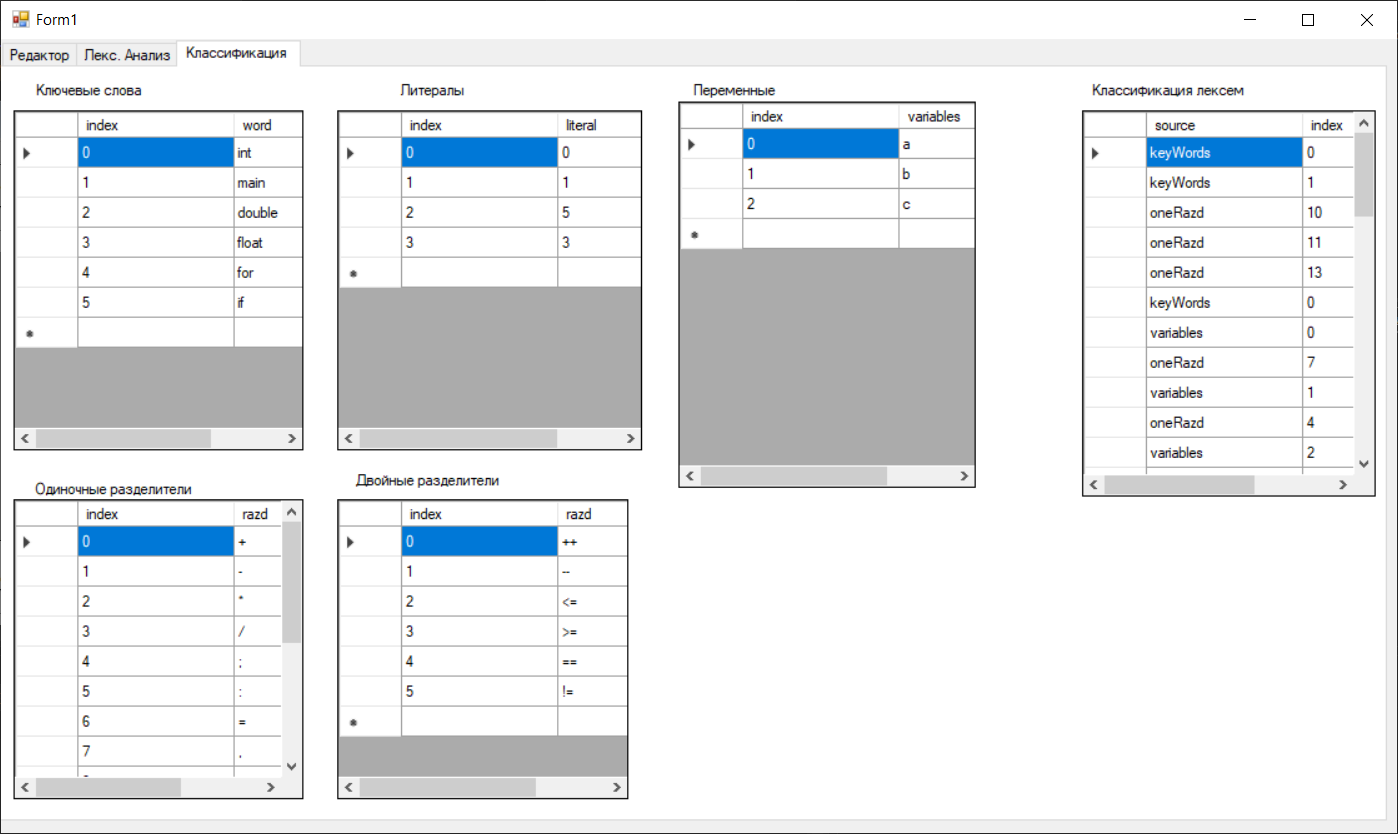


Рисунок 3 – Классифицированные лексемы

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 может быть программно реализован восходящий анализатор 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 c.

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 c.

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

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/TomorrowSolutions/TheoriaAvotmatovKurs