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

*Курсовая работа*

По дисциплине Теория автоматов и формальных языков

Тема Транслятор с подмножества языка C

Руководитель

Кульков Я.Ю

(фамилия, инициалы)

(подпись) (дата)

Студент ПИН - 121

(группа)

Антипин Д. Д.

(фамилия, инициалы)

(подпись) (дата)

Муром 2023

Целью данной курсовой работы является разработка транслятора с подмножества языка C Данная программа будет реализовано на языке программирования C# в среде разработки «Microsoft Visual Studio 2019».

Пользовательский интерфейс программы реализован с помощью Windows Forms.

The purpose of this coursework is to develop a translator from a subset of the C language. This program will be implemented in the C# programming language in the Microsoft Visual Studio 2019 development environment.

The user interface of the program is implemented using Windows Forms.

Содержание

[Введение 5](#_Toc136067670)

[1. Анализ технического задания 6](#_Toc136067671)

[2. Описание грамматики языка 7](#_Toc136067672)

[3. Разработка архитектуры системы и алгоритмов 20](#_Toc136067673)

[3.1 Описание работы лексического анализатора 20](#_Toc136067674)

[3.2 Описание работы сканера 20](#_Toc136067675)

[3.3 Описание работы синтаксического анализатора 21](#_Toc136067676)

[4. Тестирование 22](#_Toc136067677)

[5 Руководство пользователя 24](#_Toc136067678)

[6. Руководство программиста 26](#_Toc136067679)

[Заключение 27](#_Toc136067680)

[Список литературы 28](#_Toc136067681)

### Введение

Программа, написанная на языке высокого уровня, перед исполнением должна быть преобразована в программу на "машинном языке". Такой процесс называется трансляцией, или компиляцией. Для исходной программы в машинный код используется транслятор.

Транслятор (англ. translator — переводчик) — это программа-переводчик. Она преобразует программу, написанную на одном из языков высокого уровня, в программу, состоящую из машинных команд.

Исходными данными для работы транслятора служит текст входной программы – некоторая последовательность предложений входного языка программирования, удовлетворяющая синтаксическим требованиям.

Программу, написанную на языке программирования высокого уровня, называют исходной программой, а каждую самостоятельную программную единицу, образующую данную программу, - программным модулем. Для преобразования исходной программы в ее выполняемую форму (выполнимый файл) транслятор выполняет некоторую последовательность действий. Эта последовательность зависит как от языка программирования, так и от конкретной реализации самого транслятора.

В процессе трансляции выполняется анализ исходной программы, а затем синтез выполнимой формы данной программы. В зависимости от числа просмотров исходной программы, выполняемых компилятором, трансляторы разделяются на однопроходные, двухпроходные и трансляторы, использующие более двух проходов.

На основе проведенного анализа, актуальность темы работы находится на этапе учебных целей. Разбор реализации лексического, синтаксического анализа, сложного арифметического выражения, даёт возможность понять как устроен транслятор подмножества для языка программирования на стороне разработчика.

### Анализ технического задания

В представленной курсовой работе необходимо спроектировать транслятор подмножества языка C.

Анализируя тему данной курсовой работы, требуется, чтобы в созданной программе присутствовали:

1. Язык для трансляции – С;
2. Развернутая диагностика ошибок.
3. Реализация класса транслятора.
4. Синтаксический разбор – на основе LR(k)-грамматик.
5. Разбор выражений, выполненный методом Дейкстры.

В языке поддерживаются:

1. Идентификаторы, значащие первые 8 символов.
2. Не менее трех директив описания переменных.
3. Сложный арифметический оператор.
4. Простое логическое выражение
5. Оператор цикла for(…) {..}

Представленная курсовая работа реализуется в несколько шагов:

Создание лексического анализа, который в свою очередь выполняет анализ полученных данных, а также распознает лексемы и их типы.

Полученная информация обрабатывается на основе синтаксического анализа.

Синтаксический анализ обрабатывает данные, полученные в ходе работы лексического анализа, посредством нахождения синтаксических выражений и конструкций.

### Описание грамматики языка

Программа, написанная на языке C, состоит из операторов. Каждый оператор вызывает выполнение некоторых действий на соответствующем шаге выполнения программы.

Как и любой другой язык, язык C имеет свой алфавит, в который включены 32 буквы латинского алфавита, цифры от 0 до 9, арифметические операции (+ , - , \*, /, %), знаки отношений (<, >, ==, !=), разделительные знаки [:, , ; ‘ ,“ , ( ), {}, |, / ].

Важным понятием языка является идентификатор, который используется в качестве имени объекта (функции, переменной, константы и др.).

Идентификаторы должны выбираться с учетом следующих правил:

1. Они должны начинаться с буквы латинского алфавита (а,...,z, А,...,Z) или с символа подчеркивания (\_).
2. В них могут использоваться буквы латинского алфавита, символ подчеркивания и цифры (0,...,9). Использование других символов в идентификаторах запрещено.
3. В языке Си буквы нижнего регистра (а,...,z), применяемые в идентификаторах, отличаются от букв верхнего регистра (А,...,Z). Это означает, что следующие идентификаторы считаются разными: name, NaMe, NAME и т.д.
4. Идентификаторы могут иметь любую длину, но воспринимается и используется для различения объектов (функций, переменных, констант и т.д.) только часть символов. Их число меняется для разных систем программирования, но в соответствии со стандартом ANSI C не превышает 32 (в Си++ это ограничение снято). Если длина идентификатора установлена равной 5, то имена count и counter будут идентичны, поскольку у них совпадают первые пять символов.
5. Идентификаторы для новых объектов не должны совпадать с ключевыми словами языка и именами стандартных функций из библиотеки.

Учитывая все вышеперечисленные правила, в ходе работы создана грамматика языка.

G= {T, N, P, <программа>}

T = {Main, int, for, =, \*, +, <, ;, {, }, , , string, char }

N = {<спис. объяв.>, <спис. опер>, <спис. операнд.>, <спис. перем.>, <id>, <присв>, <цикл>, <тип>, <знак>, <программа>, <объявление>, <операнд>, <expr> }

P = {

<Программа> ::= Main(){<спис. опер.>}

<тип>::= int | string | char

<спис. объяв.> ::= <тип> <объявление.> | <тип> <объявление.> <спис. объяв>

<объявление.> ::= id; | id, <обьяв>

<спис. опер.> ::= <опер.> | <опер> <спис. опер.>

<присв.> ::= id=expr;

< опер> ::= <присв.> | <цикл> | <спис.объяв>

<дв.опер>::= id<дв.знак>

<дв.знак>::= ++ | --

<операнд>::=lit | id

<цикл> ::= for (id=<операнд>; <операнд><знак> <операнд>; <дв. опер.>){ <спис. опер.>}

<знак>::= + | \* | - | / | < | = | >

Таблица 1 - Пример формирования цепочки вывода

|  |  |
| --- | --- |
| Анализируемый фрагмент программы | Полученный синтаксический вывод |
| Main() {  Int a, b;  C = 0;  For(b=1; b<5; b++) {  B=5;  C= c+b\*3;}  } | Main(){  int id, id;  id=expr;  for (id= lit; id < lit; id++)  { id=expr; id=expr;}  } |

Имея данную грамматику языка, произведём поиск леворекурсивных правил и правил с левой факторизацией.

1. Поиск леворекурсивных правил.

В построенной грамматике языка, правила содержащие левую рекурсию отсутствуют.

1. Поиск правил с левой факторизацией.
2. <спис. объяв.> ::= <тип> <объявление.> | <тип> <объявление.> <спис. объяв>

Избавляемся от левой факторизации:

<спис. объяв.> ::= тип> <объявление.> <доп. объяв>

<доп. объяв> :: = Ɛ | <спис. объяв>

1. <объявление.> ::= id; | id, <обьяв>

Избавляемся от левой факторизации:

<объявление.> ::= id; <доп. объявление>

<доп. объявление> ::= Ɛ | <спис. объявление>

1. <спис. опер.> ::= <опер.> | <опер> <спис. опер.>

Избавляемся от левой факторизации:

<спис. опер.> ::= <опер.> <доп.опер>

<доп.опер> ::= Ɛ | <спис. опер>

Полученная грамматика:

G= {T, N, P, <программа>}

T = {Main, int, for, =, \*, +, <, ;, {, }, , , string, char }

N = {<спис. объяв.>, <спис. опер>, <спис. операнд.>, <спис. перем.>, <id>, <присв>, <цикл>, <тип>, <знак>, <программа>, <объявление>, <операнд>, <expr> }

P = {

<Программа> ::= Main(){<спис. опер.>}

<тип>::= <тип2>| ε

<тип2>::= int | string | char

<спис. объяв.> ::= <тип> <объявление.>

<объявление.> ::= id; | id, <обьяв>

<спис. опер.> ::= <опер.> | <опер> <спис. опер.>

<присв.> ::= <тип>id=expr;

< опер> ::= <присв.> | <цикл> | <тип2><спис.объяв>

<дв.опер>::= id<дв.знак>

<дв.знак>::= ++ | --

<операнд>::=lit | id

<цикл> ::= for (<тип>id=<операнд>; <операнд><знак> <операнд>; <дв. опер.>) <блок опер>

<знак>::= + | \* | - | / | < | = | > | ε

<блок опер>::= <опер> | {<спис.опер>}

Таблица 2 – Решающая таблица нисходящего анализа

|  |  |  |
| --- | --- | --- |
| Правила грамматики | FIRST1 | FIRST2 |
| <Программа> ::= Main(){<спис. Опер.>} | Main() |  |
| <тип>::= <тип2>  <тип>::= ε | Int,Char,string  id |  |
| <тип2>::= int  <тип2>::= string  <тип2>::= char | Int  String  Char |  |
| <спис. объяв.> ::= <тип> <объявление.> | Int,String,Char,  id |  |
| <объявление.> ::= id; <доп. объявление> | Id |  |
| <доп. объявление> ::= Ɛ  <доп. объявление> ::= <спис. объявление> | }  Id, for, Int,Char,string  Int, String, Char, id | Int id, string id, char id, id; |
| <спис. опер.> ::= <опер.> <доп.опер> | Id,for, Int,Char,string |  |
| <доп.опер> ::= Ɛ  <доп.опер> ::= <спис. опер> | }  Id, for, Int,Char,string |  |
| <присв.> ::= <тип>id=expr; | Int,Char,string,$,  Id |  |
| < опер> ::= <присв.>  < опер> ::= <цикл>  < опер> ::= <тип2><спис.объяв> | Id  For  Int,Char,string |  |
| <дв.опер>::= id<дв.знак> | id |  |
| <дв.знак>::= ++  <дв.знак>::= -- | +  - |  |
| <операнд>::=lit  <операнд>::= id | Lit  id |  |
| for (<тип>id=<операнд>; <операнд><знак> <операнд>; <дв. опер.>) <блок опер> | for |  |
| <знак>::= +  <знак>::= \*  <знак>::= -  <знак>::= /  <знак>::= <  <знак>::= =  <знак>::= > | +  \*-  -  / <  =  > |  |
| <блок опер>::= <опер>  <блок опер>::={<спис.опер>} | Id, For, Int,Char,string  { |  |

По данным полученным в результате построения таблицы множеств FIRST и FOLLOW было доказано, что данная грамматика принадлежит к множеству LL(2).

Восходящий разбор предназначен для построения дерева разбора. Мы можем представить себе этот процесс как "свертку" исходной строки к стартовому не терминалу грамматики. Каждый шаг свертки заключается в сопоставлении некоторой подстроки и правой части какого-то правила грамматики, затем происходит замена этой подстроки на не терминал, являющийся левой частью правила. Восходящий разбор менее интуитивно понятный, чем нисходящий, но зато позволяет разбирать больше грамматик.

Первым делом, для LR(k)-анализа, был построен граф состояний автомата(Таблица 3).

Таблица 1 – Граф состояний автомата

|  |  |  |  |
| --- | --- | --- | --- |
| Состояние | Предыдущее состояние | Правила грамматики | Переход |
| 0 | - | <Программа> ::= ►Main(){<спис. опер.>} | * 1 |
| 1 |  | <Программа> ::= Main ► (){<спис. опер.>} | * 2 |
| 2 |  | <Программа> ::= Main (►){<спис. опер.>} | * 3 |
| 3 |  | <Программа> ::= Main ()►{<спис. опер.>} | * 4 |
| 4 |  | <Программа> ::= Main (){►<спис. опер.>}  <спис. опер.> ::= ►<опер.>  <спис. опер.> ::= ►<опер> <спис. опер.>  < опер> ::= ►<присв.>  < опер> ::= ►<цикл>  < опер> ::= ►<тип2><спис.объяв>  <присв.> ::= ►<тип>id=expr;  <тип>::= ►<тип2>  <тип>::= ► ε  <тип2>::=► int  <тип2>::=► string  <тип2>::=► char  <цикл> ::= ►for (<тип>id=<операнд>; <операнд><знак> <операнд>; <дв. опер.>) <блок опер> | * 5 * 6 * 6 * 7 * 8 * 9 * 10 * 9   Х   * 11 * 12 * 13 * 14 |
| 5 |  | <Программа> ::= Main (){<спис. опер.> ►} | * 15 |
| 6 |  | <спис. опер.> ::= <опер.> ►  <спис. опер.> ::= <опер> ► <спис. опер.>  <спис. опер.> ::= ►<опер.>  <спис. опер.> ::= ►<опер> <спис. опер.> | X   * 16 * 6 * 6 |
| 7 |  | < опер> ::= <присв.> ► | X |
| 8 |  | < опер> ::= <цикл>► | X |
| 9 |  | < опер> ::= <тип2>►<спис.объявлений>  <тип>::= <тип2>►  <спис. объяв.> ::= ►<тип> <объявлений.>  <тип>::= ►<тип2>  <тип>::= ► ε  <тип2>::=► int  <тип2>::=► string  <тип2>::=► char | * 17   Х   * 18 * 9   Х   * 11 * 12 * 13 |
| 10 |  | <присв.> ::= <тип>►id=expr; | * 19 |
| 11 |  | <тип2>::= int ► | Х |
| 12 |  | <тип2>::= string► | X |
| 13 |  | <тип2>::= char ► | X |
| 14 |  | <цикл> ::= for ► (<тип>id=<операнд>; <операнд><знак> <операнд>; <дв. опер.>) <блок опер> | * 20 |
| 15 |  | <Программа> ::= Main (){<спис. опер.> }► | Х |
| 16 |  | <спис. опер.> ::= <опер> <спис. опер.> ► | X |
| 17 |  | < опер> ::= <тип2><спис.объявлений>► | Х |
| 18 |  | <спис. объяв.> ::= <тип>► <объявление.>  <объявление.> ::= ►id;  <объявление.> ::= ►id, <обьявление> | * 21 * 22 * 22 |
| 19 |  | <присв.> ::= <тип>id►=expr; | * 23 |
| 20 |  | <цикл> ::= for (►<тип>id=<операнд>; <операнд><знак> <операнд>; <дв. опер.>) <блок опер>  <тип>::= ►<тип2>  <тип>::= ► ε  <тип2>::=► int  <тип2>::=► string  <тип2>::=► char | * 24 * 9   Х   * 11 * 12 * 13 |
| 21 |  | <спис. объявлений.> ::= <тип> <объявлений.> ► | Х |
| 22 |  | <объявление.> ::= id►;  <объявление.> ::= id►, <обьявлений> | * 25 * 26 |
| 23 |  | <присв.> ::= <тип>id=►expr; | * 27 |
| 24 |  | <цикл> ::= for (<тип>►id=<операнд>; <операнд><знак> <операнд>; <дв. опер.>) <блок опер> | * 28 |
| 25 |  | <объявление.> ::= id; ► | Х |
| 26 |  | <объявление.> ::= id, ► <обьявление>  <объявление.> ::= id►;  <объявление.> ::= id►, <обьявление> | * 29 * 25 * 22 |
| 27 |  | <присв.> ::= <тип>id=expr►; | * 30 |
| 28 |  | <цикл> ::= for (<тип>id►=<операнд>; <операнд><знак> <операнд>; <дв. опер.>) <блок опер> | * 31 |
| 29 |  | <объявление.> ::= id, <обьявление>► | Х |
| 30 |  | <присв.> ::= <тип>id=expr; ► | Х |
| 31 |  | <цикл> ::= for (<тип>id=►<операнд>; <операнд><знак> <операнд>; <дв. опер.>) <блок опер>  <операнд>::=►lit  <операнд>::=► id | * 32 * 33 * 34 |
| 32 |  | <цикл> ::= for (<тип>id=<операнд>►; <операнд><знак> <операнд>; <дв. опер.>) <блок опер> | * 35 |
| 33 |  | <операнд>::=lit► | Х |
| 34 |  | <операнд>::= id► | Х |
| 35 |  | <цикл> ::= for (<тип>id=<операнд>; ►<операнд><знак> <операнд>; <дв. опер.>) <блок опер>  <операнд>::=►lit  <операнд>::=► id | * 36 * 33 * 34 |
| 36 |  | <цикл> ::= for (<тип>id=<операнд>; <операнд>►<знак> <операнд>; <дв. опер.>) <блок опер>  <знак>::= ►+  <знак>::= ►-  <знак>::= ►\*  <знак>::= ►/  <знак>::= ►=  <знак>::= ►>  <знак>::= ►<  <знак>::= ► ε | * 37 * 38 * 39 * 40 * 41 * 42 * 43 * 44   Х |
| 37 |  | <цикл> ::= for (<тип>id=<операнд>; <операнд><знак>► <операнд>; <дв. опер.>) <блок опер>  <операнд>::=►lit  <операнд>::=► id | * 45 * 33 * 34 |
| 38 |  | <знак>::= ►+ | Х |
| 39 |  | <знак>::= ►- | Х |
| 40 |  | <знак>::= ►\* | X |
| 41 |  | <знак>::= ►/ | X |
| 42 |  | <знак>::= ►= | X |
| 43 |  | <знак>::= >► | X |
| 44 |  | <знак>::= <► | Х |
| 45 |  | <цикл> ::= for (<тип>id=<операнд>; <операнд><знак> <операнд>►; <дв. опер.>) <блок опер> | * 46 |
| 46 |  | <цикл> ::= for (<тип>id=<операнд>; <операнд><знак> <операнд>; ►<дв. опер.>) <блок опер>  <дв.опер>::= ►id<дв.знак> | * 47 * 48 |
| 47 |  | <цикл> ::= for (<тип>id=<операнд>; <операнд><знак> <операнд>; <дв. опер.> ►) <блок опер> | * 49 |
| 48 |  | <дв.опер>::= id►<дв.знак>  <дв.знак>::=► ++  <дв.знак>::=► -- | * 50 * 51 * 52 |
| 49 |  | <цикл> ::= for (<тип>id=<операнд>; <операнд><знак> <операнд>; <дв. опер.>) ►<блок опер>  <блок опер>::=► <опер>  <блок опер>::=► {<спис.опер>} | * 53 * 54 * 55 |
| 50 |  | <дв.опер>::= id<дв.знак>► | Х |
| 51 |  | <дв.знак>::= ++► | Х |
| 52 |  | <дв.знак>::= --► | X |
| 53 |  | <цикл> ::= for (<тип>id=<операнд>; <операнд><знак> <операнд>; <дв. опер.>) <блок опер>► | X |
| 54 |  | <блок опер>::= <опер>► | X |
| 55 | 1 | <блок опер>::= {►<спис.опер>}  <спис. опер.> ::= ►<опер.>  <спис. опер.> ::= ►<опер> <спис. опер.>  < опер> ::= ►<присв.>  < опер> ::= ►<цикл>  < опер> ::= ►<тип2><спис.объяв>  <присв.> ::= ►<тип>id=expr;  <тип>::= ►<тип2>  <тип>::= ► ε  <тип2>::=► int  <тип2>::=► string  <тип2>::=► char  <цикл> ::= ►for (<тип>id=<операнд>; <операнд><знак> <операнд>; <дв. опер.>) <блок опер> | * 56 * 6 * 6 * 7 * 8 * 9 * 10 * 9 * 11 * 12 * 13 * 14 |
| 56 |  | <блок опер>::= {<спис.опер>►} | * 57 |
| 57 |  | <блок опер>::= {<спис.опер>}► | Х |

На основе таблицы 1, сформируем решающую таблицу восходящего анализатора, для разработки логики работы транслятора.

Таблица 2 – Решающая таблица LR-анализатора

|  |  |  |  |
| --- | --- | --- | --- |
| Состояние | Стек разбора | Вход | Действие |
| 0 | <Программа>  Main |  | Сдвиг  S1 |
| 1 | Main  ( |  | Сдвиг  S2 |
| 2 | (  ) |  | Сдвиг  S3 |
| 3 | )  { |  | Сдвиг  S4 |
| 4 | {  <спис опер>  <опер>  <присв>  <цикл>  <тип2>  <тип>  Int  String  Char  For |  | Сдвиг  S5  S6  S7  S8  S9  S10  S11  S12  S13  S14 |
| 5 | <спис опер>  } |  | Сдвиг  S15 |
| 6 | <опер>  <спис. опер>  <опер> |  | Свертка (-1,<спис. опер>)  S16  S6 |
| 7 | <присв> |  | Свертка(-1,<опер>) |
| 8 | <цикл> |  | Свертка(-1,<опер>) |
| 9 | <тип2>  <спис обьявлений>  <тип2>  <тип>  Int  String  Char |  | Сдвиг  S17  Свертка(-1<тип>)  S18  S11  S12  S13 |
| 10 | <тип>  Id |  | Сдвиг  S19 |
| 11 | Int |  | Свертка (-1, <тип2>) |
| 12 | String |  | Свертка (-1, <тип2>) |
| 13 | Char |  | Свертка (-1, <тип2>) |
| 14 | For  ( |  | Сдвиг  S20 |
| 15 | } |  | Свертка (-6, <программа>) |
| 16 | <спис опер> |  | Свертка (-2, <спис.опер>) |
| 17 | <спис обьявлений> |  | Свертка (-2,<опер>) |
| 18 | <тип>  <обьявление>  id |  | Сдвиг  S21  S22 |
| 19 | Id  = |  | Сдвиг  S23 |
| 20 | (  <тип>  <тип2>  Int  String  Char |  | Сдвиг  S24  S9  S11  S12  S13 |
| 21 | <обьявление> |  | Свертка (-2, <спис.обявлений>) |
| 22 | Id  ;  , |  | Сдвиг  S25  S26 |
| 23 | =  expr |  | Сдвиг  S27 |
| 24 | <тип>  id |  | Сдвиг  S28 |
| 25 | ; |  | Свертка(-2, <обьявление>) |
| 26 | ,  <обьявление>  ;  , |  | Сдвиг  S29  S25  S22 |
| 27 | Expr  ; |  | Сдвиг  S30 |
| 28 | Id  = |  | Сдвиг  S31 |
| 29 | <обьявление> |  | Свертка(-3,<обьявление>) |
| 30 | ; |  | Свертка(-5,<присв>) |
| 31 | =  <операнд>  Lit  id |  | Сдвиг  S32  S33  S34 |
| 32 | <операнд>  ; |  | Сдвиг  S35 |
| 33 | lit |  | Свертка (-1,<операнд>) |
| 34 | id |  | Свертка (-1, <операнд>) |
| 35 | ;  <операнд>  Lit  id |  | Сдвиг  S36  S33  S34 |
| 36 | <операнл>  <знак>  +  -  \*  =  /  >  < |  | Сдвиг  S37  S38  S39  S40  S41  S42  S43  S44 |
| 37 | <знак>  <операнд>  Lit  id |  | Сдвиг  S45  S33  S34 |
| 38 | + |  | Свертка (-1,<знак>) |
| 39 | - |  | Свертка (-1,<знак>) |
| 40 | \* |  | Свертка (-1,<знак>) |
| 41 | = |  | Свертка (-1,<знак>) |
| 42 | / |  | Свертка (-1,<знак>) |
| 43 | > |  | Свертка (-1,<знак>) |
| 44 | < |  | Свертка (-1,<знак>) |
| 45 | <операнд>  ; |  | Сдвиг  S46 |
| 46 | ;  <дв.опер>  id |  | Сдвиг  S47  S48 |
| 47 | <дв.опер>  ) |  | Сдвиг  S49 |
| 48 | Id  <дв.знак>  ++  -- |  | Сдвиг  S50  S51  S52 |
| 49 | )  <блок.опер>  <опер>  { |  | Сдвиг  S53  S54  S55 |
| 50 | <дв.знак> |  | Свертка(-2,<дв.опер>) |
| 51 | ++ |  | Свертка (-2, <дв.знак>) |
| 52 | -- |  | Свертка (-2, <дв.знак>) |
| 53 | <блок.опер> |  | Свертка (-14, <цикл>) |
| 54 | <опер> |  | Свертка (-1, <блок.опер>) |
| 55 | {  <спис.опер>  <опер>  <присв>  <цикл>  <тип2>  <тип>  Int  String  Char  For |  | Сдвиг  S56  S6  S7  S8  S9  S10  S11  S12  S13  S14 |
| 56 | <спис.опер>  } |  | Сдвиг  S57 |
| 57 | } |  | Свертка(-3, <блокюопер>) |

### Разработка архитектуры системы и алгоритмов

Для реализации транслятора с подмножеством для языка C требуется разработать алгоритм лексического, синтаксического анализатора, а также разбор сложного арифметического выражения.

### 3.1 Описание работы лексического анализатора

1. На вход алгоритм получает исходную строку, которую в дальнейшем считывает посимвольно.
2. До тех пор, пока не будет достигнут конец входной строки или получена ошибка лексического анализа, считываем очередной символ анализируемой строки, определяем и переходим из текущего состояния диаграммы в другое, выполняя при этом соответствующие действия. Состояние, в которое попадаем, на данный момент, становится текущим.
3. Выходной строка с лексемой формируется в состояния, где выходная строка удовлетворяет определённым условиям. Тип лексемы изменятся в зависимости от текущего состояния. В каждом состояние выходная лексема проверяется на наличие ошибок, если встретилась ошибка – работа лексического анализатора прекращается. При успешном проведенном лексическом анализе, анализатор вернёт список всех найденных лексем, в виде отношения: Лексема – тип. Для первого этапа разбора предварительные типы лексем: идентификатор, разделитель, литерал

### 3.2 Описание работы сканера

Работа сканера заключается в моделировании различных конечных автоматов для распознавания идентификаторов, зарезервированных слов, констант и разделителей.

В процессе получения на вход символа, цикл производит проверку.

Если символ не является ни буквой, и не цифрой, следовательно, цикл присваивает ему значение «Идентификатор». В случае, если на вход получена цифра, анализатор классифицирует ее как «Литерал». Если на вход получен символ «\n», программа инициализирует ее как «Конец строки». В противном случае присваивается значение «Разделитель».

Результатом работы сканера является последовательность кодов лексем. Эту последовательность обычно называют таблицей стандартных символов, так как в ней хранятся стандартизованные представления лексем. Информация в этой таблице расположена в том же порядке, что и в исходной программе.

Пример работы сканера представлен в таблице ниже:

Таблица 3-Пример работы сканера

|  |  |
| --- | --- |
| Main | Идентификатор |
| () | Конец строки |
| { | Разделитель |
| int | Идентификатор |
| a | Идентификатор |

### 3.3 Описание работы синтаксического анализатора

Синтаксический анализ является частью компилятора, которая взаимодействует с синтаксическими конструкциями языка, с помощью токенов.

Токен – некая структура данных, которая состоит из имени и набора необязательных произвольных атрибутов. Имя токена представляет собой абстрактный символ, который в свою очередь представляет тип лексической единицы, например, <ключевое слово>, <название переменной>, и т.п.

Синтаксический анализ, взаимодействующий с синтаксическими конструкциями языка с помощью токенов, является частью компилятора.

Лексический анализ использует токены для токенизации, то есть классификации разделов строки входных символов. Большинство методов анализа данных используют нисходящие или восходящие алгоритмы.

### Тестирование

Проверка лексического анализатора осуществляется путём ввода корректного текста на языке C. Ввод корректных лексем, заполнит пустые таблицы приложения данными как на рисунке 1-2.

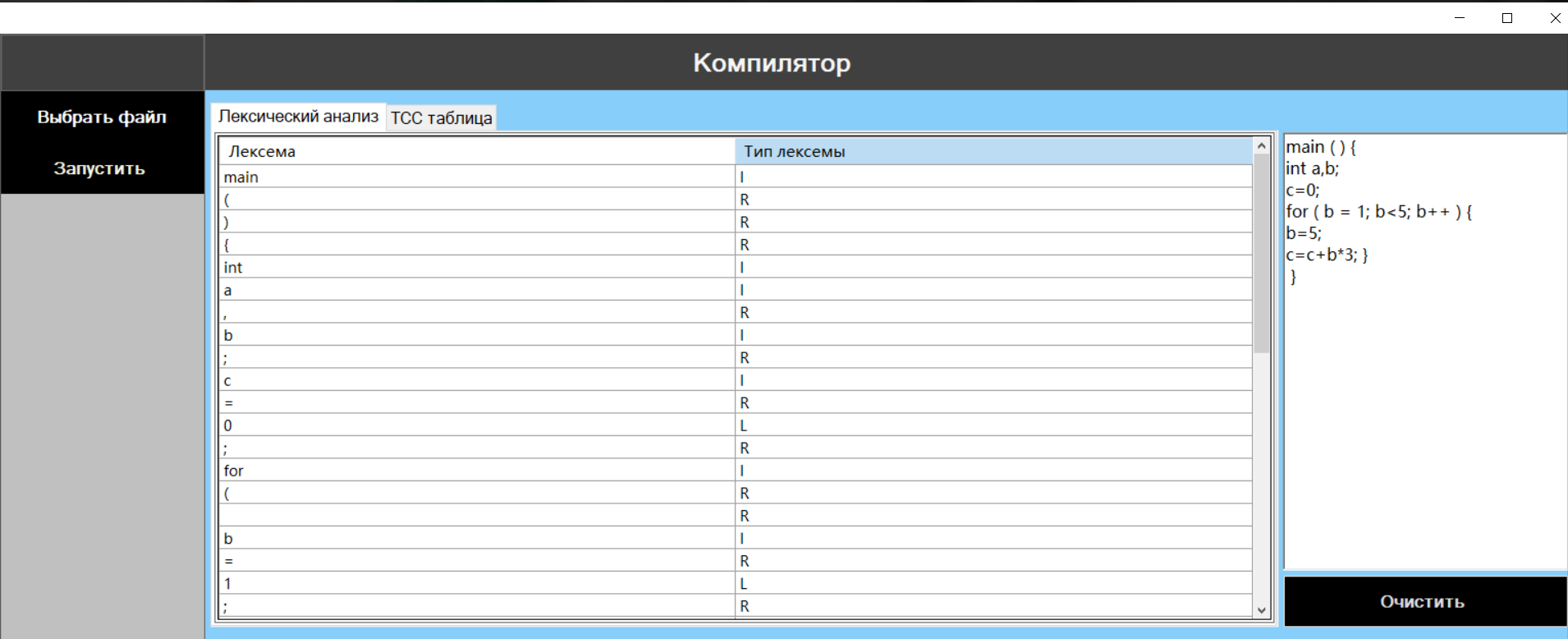


Рисунок 1 – Результат первого этапа лексического анализатора

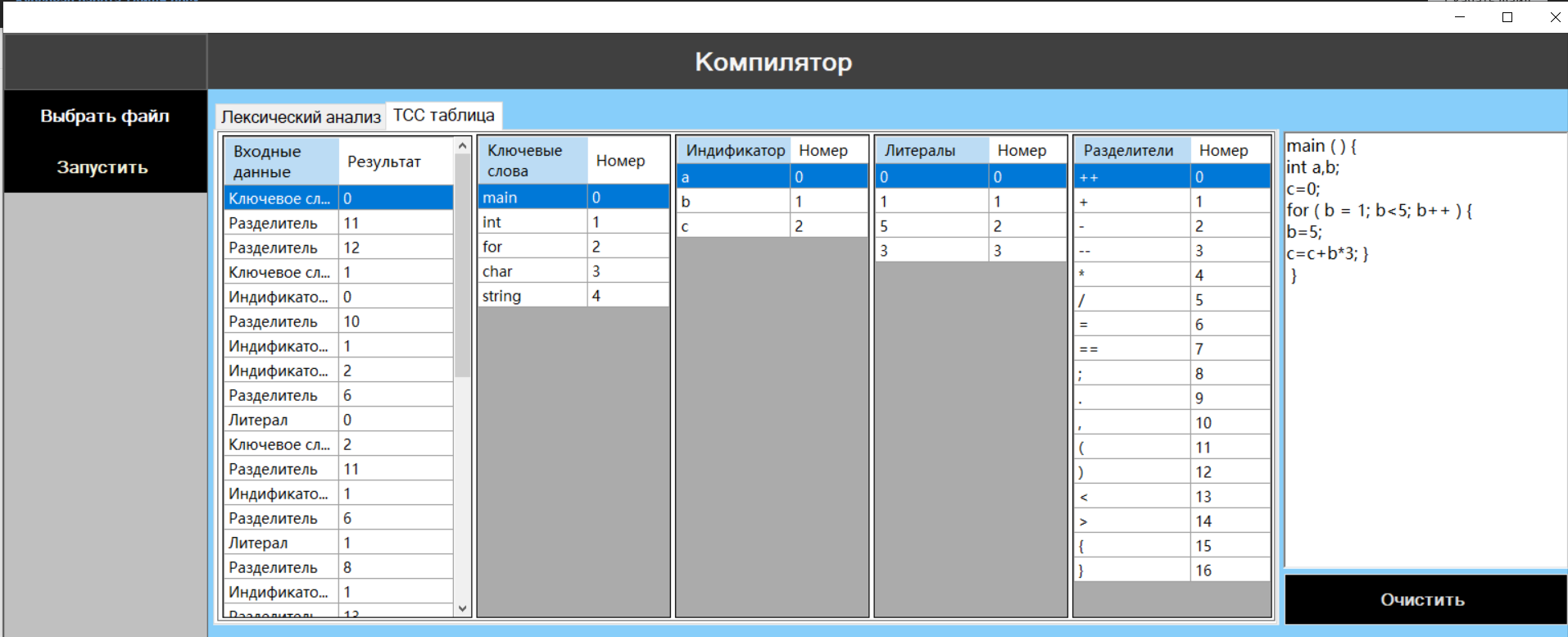


Рисунок 2 – Результат работы второго этапа лексического анализатора

При вводе символа, неразрешённого грамматикой языка, выводится сообщение с ошибкой в виде диалогового окна (рисунок 3).

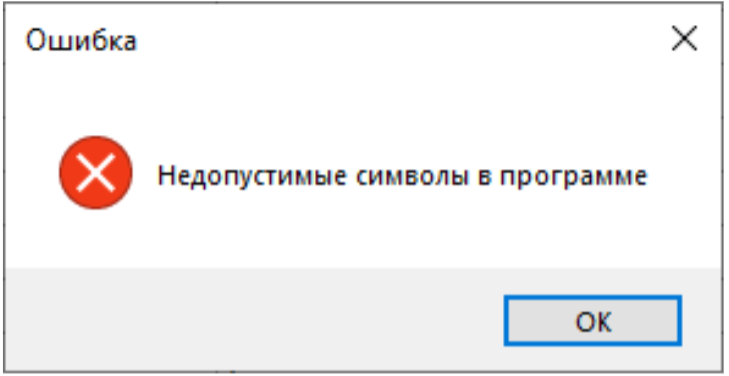


Рисунок 3 – Диалоговое окно с ошибкой

При вводе цифры перед ключевым словом аналогично выводится сообщение с ошибкой в виде диалогового окна (рисунок ).

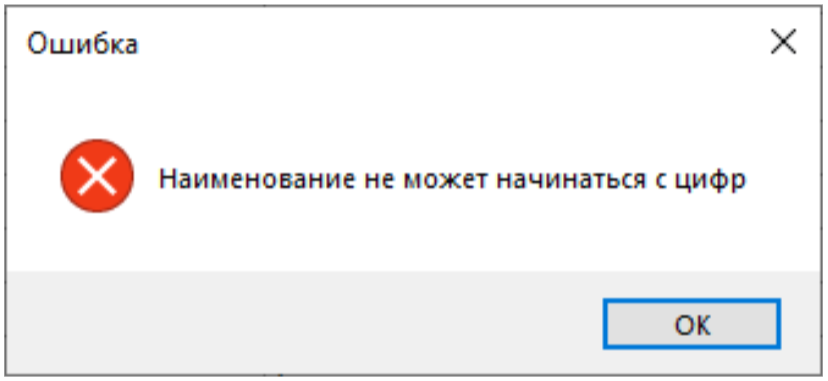


Рисунок 4 – Диалоговое окно с ошибкой

### Руководство пользователя

Запуск исполняемого файла приложения откроет главное окно с программой (рисунок 5). Пользователь может взаимодействовать с приложением благодаря кнопкам «Выбрать файл», «Запустить». Кнопка «Запустить» запустит транслятор и пользователь увидит заполненные таблицы.

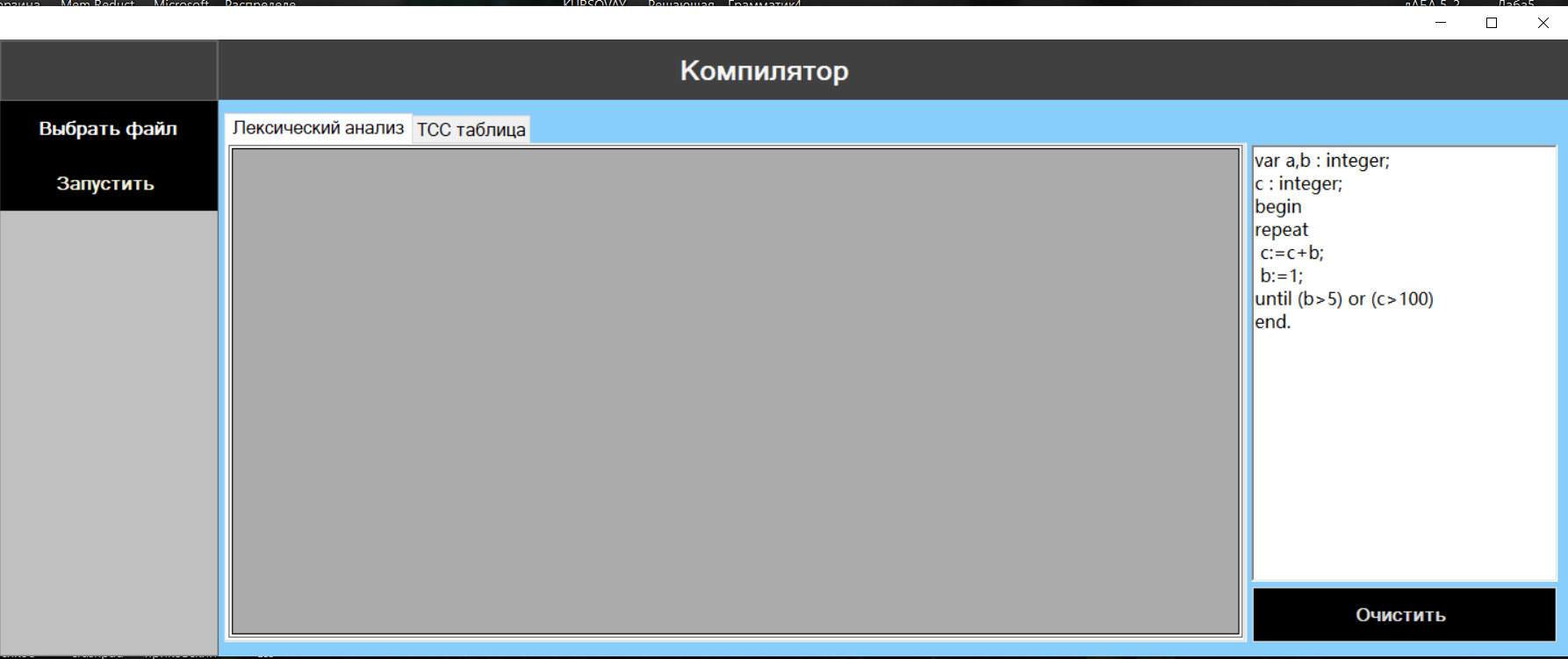


Рисунок 5 – Интерфейс приложения

Для того чтобы посмотреть вторую часть лексического анализа следует нажать на вкладку «ТСС таблица» (рисунок 6).

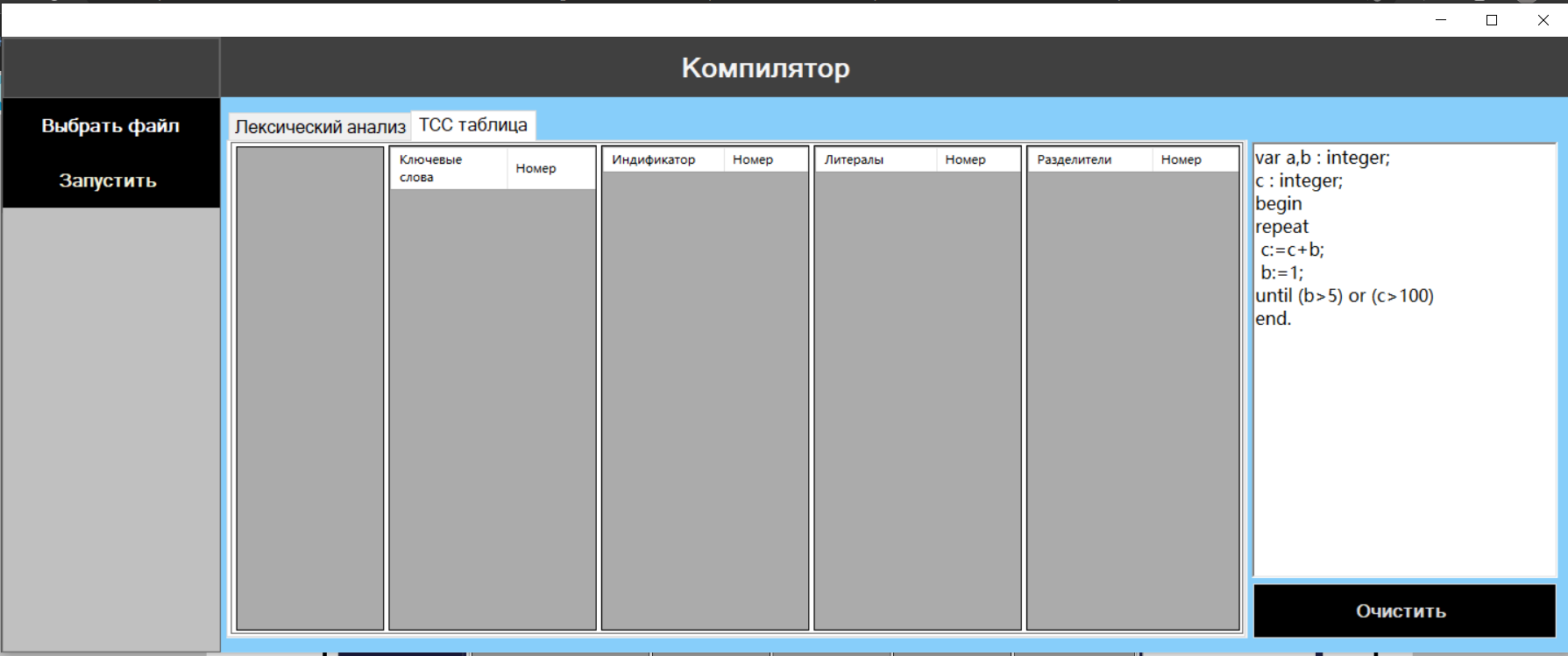


Рисунок 6 – Интерфейс приложения

Кнопка «Выбрать файл» откроет новое диалоговое окно, где пользователь может считать код из файла формата «.txt» (рисунок 7).

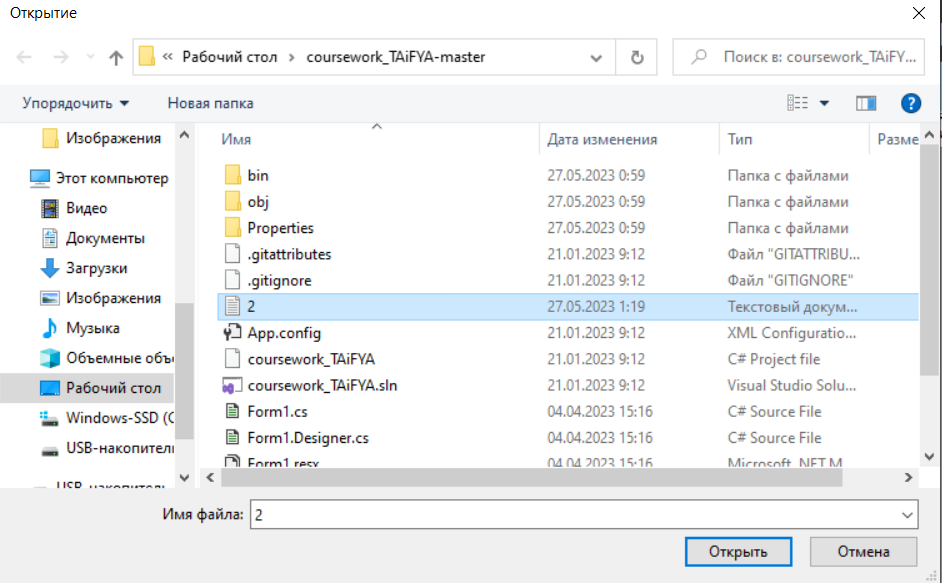


Рисунок 7 – Диалоговое окно выбора «.txt» файла

### Заключение

На основе проведенного анализа технического задания, в ходе курсовой работы был разработан транслятор с подмножества языка C, соответствующий всем требования технического задания. Приложение реализовано в среде разработки Visual Studio при помощи типизированного, объектно-ориентированного языка программирования C#.

Разработанное приложение производит лексический анализ. Над разработанным приложением проведены тесты, результаты тестирования соответствуют ожидаемым результатам.

Программа разработана для операционной системы Windows и не требует дополнительного программного обеспечения для работы

### Список литературы

1. Малявко, А. А. Формальные языки и компиляторы: учебник / А. А. Малявко. — Новосибирск: Новосибирский государственный технический университет, 2014. — 431 c.
2. 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.
3. Карпов В.Э. К26 Теория компиляторов. Учебное пособие. 2-е изд., испр. и дополн. М., 2018. – 92 с.
4. 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.