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 Теория автоматов и формальных языков

Тема: Транслятор с подмножества языка Visual Basic

Руководитель:

Кульков Я.Ю.
(фамилия, инициалы)

(подпись) (дата)

Студент ПИн - 121
(группа)

Маресев С. А.
(фамилия, инициалы)

(подпись) (дата)

Муром 2023

В данной курсовой работе необходимо было спроектировать транслятор подмножества языка Visual Basic. В качестве средств разработки приложения была использована среда Microsoft Visual Studio 2022. Язык разработки: C#.

In this course work, it was necessary to design a translator for a subset of the Visual Basic language. The Microsoft Visual Studio 2022 environment was used as the application development tools. Development language: C#.

Содержание

Введение

С появлением первых высокоуровневых языков программирования (приближенных к человеческому языку) необходимо было создать программы, которые будут переводить программы в двоичный машинный код с которым работает компьютер. Данными программами стали трансляторы.

Транслятор – это программа или часть программного обеспечения, производящая трансляцию полученной программы. Транслятор преобразует программу написанную на одном из языков программирования, в программу, написанную на другом языке. Также транслятор осуществляет диагностику программы на ошибки, формирует словари лексем и их классификации.

Язык, на котором была представлена программа называется исходным языком, а код программы исходным кодом. То есть исходными данными транслятора будет текст программы, соответствующий синтаксическим требованиям исходного языка.

Все трансляторы состоят из следующих этапов:

- Лексический анализ/разбор – определение лексем из исходного кода по 4 типам: Ключевые слова языка, разделители, идентификаторы (Названия переменных) и литералы (Числовые переменные).
- Синтаксический анализ/разбор на основе LL или LR грамматик выполняет выделение синтаксических конструкций в тексте исходной
- программы, обработанной лексическим анализатором. На этом же этапе проверяется синтаксическая правильность программы.
- Разбор сложных арифметических и сложных логических выражений методами Дейкстры или Бауэра Замельзона.

Актуальность данной темы заключается в получении практических и теоретических навыков работы с транслятором подмножества языка Visual Basic. Для успешного выполнения этой работы необходимо изучить работу синтаксического и нисходящего анализатора, понять, как создаются и проектируются базы данных, научиться также работать и оперировать с гигантскими объемами данных, чтобы при этом, выполняемые запросы выполнялись с достаточно высокой скоростью.

Из этого можно сделать вывод, что использование современных информационных систем позволяют: работать с большими объемами данных; хранить эти данные в течение длительного временного периода; связывать несколько компонентов, которые имеют свои определенные локальные цели, задачи и разнообразные приемы функционирования, в одну систему для работы с информацией и существенно снизить затраты на доступ и хранение к любым необходимым нам данным.

1. Анализ технического задания

В представленной курсовой работе необходимо спроектировать транслятор подмножества языка Visual Basic.

Требования к созданной грамматике:

– Обеспечить развернутую диагностику ошибок;

– Реализовать класс транслятора;

– Синтаксический разбор - на основе LR(k)-грамматик;

– Разбор арифметических выражений выполнять методом Дейкстры;

В языке должны поддерживаться:

– у идентификатора 8 символов значащие;

– не менее 3х директив описания переменных;

– простой арифметический оператор;

– сложное логическое выражение;

– 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.

2. Описание грамматики языка

Для краткости название языка сократим до VB. Базовые символы языка VB - специализированные символы, цифры, а также буквы. Эти наборы символов представляют алфавит языка VB.

Как и любой другой язык, язык VB имеет свой алфавит, в который включены 26 букв латинского алфавита, цифры от 0 до 9, арифметические операции (+ , - , *, /, \, ^), знаки отношений (<, >, =), разделительные знаки [: , ; ‘ “ . ( )] , другие знаки (# , $, @ , ! , ? , & , _ , %.). Из алфавита данного языка можно сложить разнообразные конструкции языка, такие как: данные; операторы, выражения и функции.

Для разбора математических выражений с множественными операторами и скобками был использован алгоритм Дейкстры вместо синтаксического анализатора. Основываясь на правилах написания операторов, можно составить грамматику данного подмножества языка VB.

Имена переменных в VB подчиняются ряду ограничений :

1. Первым символом имени должна быть *буква.*

2. Остальные символы - *буквы* и *цифры.* (Прописные и строчные буквы различаются.)

3. Можно использовать знак _. Нельзя использовать точку.

4. Число символов не должно превышать 255.

5. Имя не должно быть *ключевым словом* VB.

В языке VB версии 6.0 переменная может иметь один из более чем десяти типов. Учитывая все вышеперечисленные правила, в ходе работы создана грамматика языка.

Служебными словами в данном подмножестве являются: Dim, as, integer, string, bool, for, to, next.

В курсовой работе предусматривается что математическое выражение будет иметь сложную структуру, то есть иметь множественные операторы и действия, следовательно разбор такого выражения целесообразно выполнять

другим алгоритмом. Вместо синтаксического анализатора может использоваться алгоритм Дейкстры.

Исходя из правил написания операторов языка, можно составить грамматику данного подмножества языка Visual Basic.

Полученная грамматика:

G= T,N,P,<программа>

T= {Dim, as, integer, bool, string, for, to, next, id, lit, expr, =, +, -, *, /, \n}

N={<программа>, <объяв.>, <спис.опер.>, <тип>, <опер>, <условн.>, <присвоен.>, <иниц>, <знак>, <слож.операнд>, <операнд>, <спис.опер.>, <спис.идент.>}

P =<программа>::= <Объяв.> \n <спис.опер.>

<объяв.>::= Dim <спис.идент.> as <тип> \n

<спис.идент>::= id | id, <спис.идент.>

<тип>::= integer | bool | string

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

<иниц>::= Dim <спис.идент.> as <тип>

<опер>::= <присвоен.> | <условн.> | <иниц>

<присвоен.>::= id=epxr

<слож.операнд>::= id=<операнд>

<операнд>::= id | lit

<условн.>::= for <слож.операнд> to <операнд> \n <спис.опер.> next for

Пример цепочки вывода:

<программа> ::= <объяв.> \n <спис.опер> =>
Dim <спис.идент.> as <тип> \n <спис.опер> =>
Dim id as integer \n <опер> \n <спис.опер> =>
Dim id as integer \n <опер> \n <опер> \n <спис.опер> =>
Dim id as integer \n <опер> \n <опер> =>
Dim id as integer \n <иниц> \n <условн.> =>
Dim id as integer \n Dim <спис.идент.> as <тип> \n for <слож.операнд> to <операнд> \n <спис.опер.> \n next for =>
Dim id as integer \n Dim id, <спис.идент.> as <тип> \n for id=<операнд> to lit \n <спис.опер.> \n next for =>
Dim id as integer \n Dim id, id as integer \n for id=lit to lit \n <опер> \n next for =>
Dim id as integer \n Dim id, id as integer \n for id=lit to lit \n <присвоен.> \n next for =>
Dim id as integer \n Dim id, id as integer \n for id=lit to lit \n id=epxr \n next for

Таблица 1 - Пример итогового формирования цепочки вывода

Анализируемый фрагмент программы	Полученный синтаксический вывод
Dim x as integer Dim i, a as integer for i=1 to 10 x=x+i*5 next for	Dim id as integer Dim id, id as integer for id=lit to lit id=expr next for

3. Разработка архитектуры системы и алгоритмов

Во процессе работы с курсовым проектом был создан лексический анализатор. Лексический анализатор является частью компилятора, которая считывает литералы программы и строит из них лексемы. Лексический анализ обрабатывает исходный текст, полученный от пользователя, и распознает лексемы, а также классифицирует их.

Лексический анализатор выделяет из текста лексемы различных типов: идентификаторы, литералы (числовые и символьные константы), разделители. Выделение (сборка) лексемы сопровождается проверки её правильности. Обнаруженные лексические ошибки фиксируются. Язык описания лексических единиц в большинстве случаев является регулярным, то есть может быть описан с помощью регулярных грамматик.

Распознавателями регулярных языков являются конечные автоматы. Одним из способов описания конечного автомата является графическое его представление в виде маркированного однонаправленного графа, в котором узлы соответствуют состояниям конечного автомата, дуги отображают переходы из одного состояния в другое, а символы маркировки дуг соответствуют функции перехода конечного автомата.

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». В случае получения знака 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 на представлен на таблице 2 ниже:

Таблица 2 - Пример работы сканера

Dim	Идентификатор
=	Разделитель
5	Литерал
next	Идентификатор
,	Разделитель

При описании лексического анализа, важно понимать и разделять три связанных понятия.

Токен - структура, состоящая из имени и набора необязательных произвольных атрибутов. Имя токена – абстрактный символ, представляющий тип лексической единицы, например, <ключевое слово>, <название переменной>, и т.п.

Лексема - представляет собой последовательность символов исходной программы, которая идентифицируется лексическим анализатором как экземпляр токена.

Шаблон - описание вида, который может принимать лексема токена. Лексический анализатор принимает решение о принадлежности лексемы токену на основе шаблона.

Лексический токен (или просто токен) – это строка с присвоенным и, таким образом, идентифицированным значением. Он структурирован как пара, состоящая из имени токена и необязательного значения токена. Имя токена — это категория лексической единицы.

На вход лексического анализатора поступает текст исходной программы, а выходная информация передаётся для дальнейшей обработки синтаксическому анализатору. Каждой выделенной из текста лексеме сканер ставит в соответствие токен вида: ‹имя_токена, значение_атрибута›. <имя_токена>, представляет собой абстрактный символ, использующийся во время синтаксического анализа, а второй компонент, значение атрибута, указывает на запись в таблице идентификаторов, соответствующую данному токену.

Токенизация — это процесс классификации разделов строки входных символов. Полученные токены затем передаются на следующий этап компиляции, то есть в синтаксический анализатор.

Итогом работы лексического анализатора должен быть список лексем, в котором каждой сопоставлен ее конкретный тип. Далее эта информация будет использована синтаксическим анализатором для проверки принадлежности грамматике. Пример работы лексического анализатора приведён в таблице 3.

Таблица 3 - Пример работы лексического анализатора

DIM	Dim
LITERAL	5
EQUAL	=
NEXT	next
COMMA	,

Синтаксическим анализом (разбором) называется процесс, определяющий порождается ли данная строка лексем данной грамматикой.

Синтаксический анализатор может быть построен для любой грамматики. Для любой контекстно свободной грамматики существует анализатор, требующий самое большее O(n3) времени для разбора строки из n лексем. Но для языков, которые встречаются на практике, хватает линейных алгоритмов. Анализаторы языков программирования почти всегда проходят один раз слева направо входной текст, считывая за раз одну лексему.

Большинство методов синтаксического разбора относятся к «восходящим» или «нисходящим» методам. В первом случае построение дерева начинается с корня и продолжается в направлении листьев, в то время как во втором – построение начинается с листьев и развивается по направлению к корню. Популярность анализаторов «сверху-вниз» объясняется тем, что эффективный анализатор можно легко построить «вручную». Анализаторы «снизу-вверх» могут работать с большим классом грамматик и схем перевода, поэтому в программах-генераторах анализаторов

всё больше используются восходящие алгоритмы разбора.

Восходящий анализ в классе LR(k)-грамматик. Восходящий синтаксический анализ для грамматики G = (T, N, P, S) начинает разбор с конкретных слов (лексем) из множества T, связывая сначала пары слов, затем подсоединяет к этим парам новые слова или другие связанные пары образуя нетерминалы из множества N. Постепенно процесс связывания доходит до начального нетерминала S - то есть все лексемы оказываются связаны в единую структуру.

Восходящие методы синтаксического анализа состоят в том, что в цепочке (промежуточной или терминальной) ищется правая часть очередного правила, которое должно быть заменено своим нетерминалом. Т.е. синтаксическое дерево строится снизу-вверх: в текущем множества «незакрытых» вершин ищется подмножество потомков и над ними «надстраивается» вершина-предок. При этом обход вершин и, аналогичный, просмотр цепочки символов происходит слева- направо.

Задача восходящего анализа и состоит в поиске "редукции" исходной терминальной цепочки в аксиому (нетерминал), где на каждом шаге производится поиск основы в текущую цепочку с последующей заменой ее соответствующим нетерминалом, т.е. применяется "инверсия" некоторого правила данной КС - грамматики: правая часть правила заменяется левой. Нетрудно понять, что если мы заменяем на каждом шаге такой "редукции" самую левую основу, то восстанавливаем с конца к началу правый вывод исходной цепочки, а если самую правую основу, то "реконструируем" левый вывод.

Во - первых, по аналогии с нисходящим разбором можно предположить, что для обнаружения основы достаточно пары символов – последнего символа основы и следующего символа строки. Т.е. для каждой пары символов грамматики однозначно можно сформулировать утверждение, является ли эта пара концом основы или нет. Опять - таки это связано с глубиной просмотра вперед входной строки – она равна 1.

Во - вторых, распознавателю необходим стек, и для него требуется определить функциональное назначение. Поскольку просмотр строки в поисках основы требует сохранения пройденных символов, резонно это делать в стеке. Тогда замена правой части правила (основы) на левую будет также производиться в вершине стека. Сам стек будет хранить «недосвернутую» просмотренную часть цепочки, для которой еще не накоплена основа.

Теперь можно сформулировать основные принципы восходящего разбора с использованием магазинного автомата, именуемого также методом «свёртка-сдвиг»:

1) Первоначально в стек помещается первый символ входной строки, а второй становится текущим;

2) Автомат выполняет два основных действия:

– сдвиг очередного символа из входной строки в стек (с переходом к следующему);

– поиск правила, правая часть которого хранится в стеке. В таком случае производится замена ее на левую

– свёртка;

Решение, какое из действий – перенос или свёртка должно быть выполнено на данном шаге, принимается на основе анализа пары символов – символа в вершине стека и очередного символа входной строки. Свёртка соответствует наличию в стеке основы, при ее отсутствии выполняется перенос. Управляющими данными автомата является решающая таблица, содержащая для каждой пары символов грамматики указание на выполняемое действие (свёртка, сдвиг или переход в другое состояние) и сами правила грамматики.

Конфликт относится к типу сдвиг-свёртка, если для одной цепочки допустимы и сдвиг и свёртка. Конфликт относится к типу свёртка-свёртка, если допустимы свёртки по различным правилам.

Если в процессе LR-разбора принять детерминированное решение о сдвиге/свертке удаётся, рассматривая только цепочку α и первые k символов

не просмотренной части входной цепочки, говорят, что грамматика обладает LR(k) - свойством.

В LR(k) разборе учитывается k-первых символов не просмотренной части входной цепочки.

Положительным результатом работы автомата будет наличие начального нетерминала грамматики в стеке при пустой входной строке.

Построим граф состояний автомата в таблице 4 и решающую таблицу детерминированного автомата в таблице 5. На основе анализа решающей таблицы и конфликтов сделаем вывод о принадлежности грамматики к классу LR(k) и определим k.

Таблица 4 – Граф состояний автомата

Состоя-ние	Предыдущее состояние	Правила грамматики	Переход
0	-	<программа>::=∗<объяв.><спис.опер.> <объяв.>::= ∗Dim <спис.идент.> as <тип> \n	>1 >2
1	0	<программа>::=<объяв.>∗<спис.опер.> <спис.опер.>::= ∗<опер> \n <спис.опер.>::= ∗<опер> \n <спис.опер.> <опер>::= ∗<присвоен.> <опер>::= ∗<условн.> <опер>::=∗<иниц> <иниц>= ∗Dim <спис.идент.> as <тип> <присвоен.>::=∗id=epxr <условн.>::= ∗for <слож.операнд> to <операнд> \n<спис.опер.> next for	>3 >4 >4 >17 >18 >40 >41 >19 >20
2	0	<объяв>::=Dim∗<спис.идент.> as <тип> \n <спис.идент>::= ∗id <спис.идент>::= ∗id, <спис.идент.>	>5 >6 >6
3	1	<программа>::=<объяв.> <спис.опер.>∗	-
4		<спис.опер.>::= <опер>∗\n <спис.опер.>::= <опер>∗\n <спис.опер.>	>15 >15
5		<объяв.>::=Dim<спис.идент.>∗as <тип> \n	>7
6		<спис.идент>::= id∗, <спис.идент.> <спис.идент>::= id∗	>13 -

7		<объяв.>::=Dim <спис.идент.> as∗<тип> \n <тип>::= ∗integer <тип>::= ∗bool <тип>::= ∗string	>8 >9 >10 >11
8		<объяв.>::=Dim<спис.идент.>as<тип>∗\n	>12
9		<тип>::= integer∗	-
10		<тип>::= bool∗	-
11		<тип>::= string∗	-
12		<объяв.>::=Dim<спис.идент.>as<тип>\n∗	-
13		<спис.идент>::= id,∗<спис.идент.> <спис.идент>::= ∗id, <спис.идент.> <спис.идент>::= ∗id	>14 >6 >6
14		<спис.идент>::= id, <спис.идент.>∗	-
15		<спис.опер.>::= <опер>\n∗ <спис.опер.>::= <опер>\n∗<спис.опер.>	- >16
16		<спис.опер.>::= <опер>\n <спис.опер.>∗	-
17		<опер>::= <присвоен.>∗	-
18		<опер>::= <условн.>∗	-
40		<опер>::= <иниц.>∗	-
19		<присвоен.>::=id∗=epxr	>21
20		<условн.>::= for∗<слож.операнд> to <операнд> \n <спис.опер.> next for <слож.операнд>::=∗id=<операнд>	>23 >25
21		<присвоен.>::=id=∗epxr	>22
22		<присвоен.>::=id=epxr∗	-
23		<условн.>::= for <слож.операнд>∗ to \n <спис.опер.> next for	>24
24		<условн.>::= for<слож.операнд> to ∗<операнд> \n <спис.опер.> next for <операнд>::= ∗id <операнд>::= ∗lit	>34 >26 >27
25		<слож.операнд>::=id∗=<операнд>	>28
26		<операнд>::= id∗	-
27		<операнд>::= lit∗	-
28		<слож.операнд>::=id=∗<операнд> <операнд>::= ∗id <операнд>::= ∗lit	>35 >26 >27
29		<знак>::= +∗	-
30		<знак>::= -∗	-
31		<знак>::= *∗	-
32		<знак>::= /∗	-

34		<условн.>::= for<слож.операнд> to <операнд>* \n<спис.опер.> next for	>36
35		<слож.операнд>::=id=<операнд>*	-
36		<условн.>::= for <слож.операнд> to <операнд> \n*<спис.опер.> next for <спис.опер.>::= *<опер>\n <спис.опер.>::= *<опер>\n <спис.опер.> <опер>::= *<присвоен.> <опер>::= *<условн.> <опер>::= *<иниц> <иниц>= *Dim <спис.идент.> as <тип> <присвоен.>::=*id=epxr <условн.>::= *for <слож.операнд> to <операнд> \n <спис.опер.> next for	>37 >4 >4 >17 >18 >40 >41 >19 >20
37		<условн.>::= for <слож.операнд> to <операнд> \n<спис.опер.> *next for	>38
38		<условн.>::= for <слож.операнд> to <операнд> \n<спис.опер.> next* for	>39
39		<условн.>::= for <слож.операнд> to <операнд> \n<спис.опер.>next for*	-
41		<иниц>= Dim *<спис.идент.> as <тип> <спис.идент>::= *id <спис.идент>::= *id, <спис.идент.>	>42 >46 >46
42		<иниц>= Dim <спис.идент.> *as <тип>	>43
43		<иниц>= Dim <спис.идент.> as *<тип> <тип>::= *integer <тип>::= *bool <тип>::= *string	>44 >9 >10 >11
44		<иниц>= Dim <спис.идент.> as <тип>*	-
46		<спис.идент>::= id*, <спис.идент.> <спис.идент>::= id*	>47 -
47		<спис.идент>::= id,*<спис.идент.> <спис.идент>::= *id, <спис.идент.> <спис.идент>::= *id	>48 >46 >46
48		<спис.идент>::= id, <спис.идент.>*	-
49		integer	-
50		bool	-
51		string	-

Таблица 5 – решающая таблица детерминированного автомата

Состояние	Стек разбора	Вход	Действие
0	<программа> Ɛ <объяв.> Dim		Конец Сдвиг S1 S2
1	<объяв.> <спис.опер.> <опер> <присвоен.> id <условн.> for <иниц> Dim		Сдвиг S3 S4 S17 S19 S18 S20 S40 S41
2	Dim <спис.идент.> id		Сдвиг S5 S6
3	<спис.опер.>		(-2, <программа>)
4	<опер> \n		Сдвиг S15
5	<спис.идент.> as		Сдвиг S7
6	id , id	, as	Сдвиг S13 (-1. <спис.идент.>)
7	as <тип> integer bool string		Сдвиг S8 S9 S10 S11
8	<тип> /n		Сдвиг S12
9	integer		(-1, <тип>)
10	bool		(-1, <тип>)
11	string		(-1, <тип>)
12	/n		(-5, <объяв.>)
13	, <спис.идент> id		Сдвиг S14 S6
14	<спис.идент.>		(-3, <объяв.>)

15	/n /n <спис.опер>	$ for, id	(-2, <спис.опер.>) Сдвиг S16
16	<спис.опер>		(-3,<опер>)
17	<присвоен.>		(-1, <опер.>)
18	<условн.>		(-1, <опер>)
40	<иниц>		(-1, <опер>)
19	id =		Сдвиг S21
20	for <слож.операнд> id		Сдвиг S23 S25
21	= expr		Сдвиг S22
22	expr		(-3, <присвоен.>)
23	<слож.операнд> to		Сдвиг S24
24	to <операнд>		Сдвиг S34
25	id =		Сдвиг S28
26	id		(-1, <операнд.>)
27	lit		(-1, <операнд>)
28	= <операнд> id lit		Сдвиг S35 S26 S27
29	+		(-1, <знак>)
30	-		(-1, <знак>)
31	*		(-1, <знак>)
32	/		(-1, <знак>)
34	<операнд> \n		Сдвиг S36
35	<операнд>		(-3, <слож.операнд>)
36	\n <спис.опер.> <опер> <присвоен.> id <условн.> for <иниц> Dim		Сдвиг S37 S4 S17 S19 S18 S20 S40 S41

37	<спис.опер.> next		Сдвиг S38
38	next for		Сдвиг S39
39	for		(-8, <опер>)
41	Dim <спис.идент.> id		Сдвиг S42 S46
42	<спис.идент.> as		Сдвиг S43
43	as <тип> integer bool string		Сдвиг S44 S49 S50 S51
44	<тип> /n		Сдвиг (-6, <иниц>)
46	id , id	, as	Сдвиг S47 (-1. <спис.идент.>)
47	, <спис.идент> id		Сдвиг S48 S46
48	<спис.идент.>		(-3, <иниц>)
49	integer		(-1, <тип>)
50	bool		(-1, <тип>)
51	string		(-1, <тип>)

Для однозначного определения основы по левому набору вхождения правой части некоторого правила вывода необходим 1-буквенный префикс правого набора символов, следовательно грамматика принадлежит к классу LR(1).

Разбор сложных арифметических выражений осуществляется методом Дейкстра.

Данный метод основан на использовании стека с приоритетами. Выражение просматривается слева направо. Операнды переписываются в выходную строку, а знаки операций помещаются вначале в стек операций. Если приоритет входного знака равен нулю или больше приоритета знака, находящегося на вершине стека, то новый знак добавляется к вершине стека. В противном случае из стека выталкивается и переписывается в выходную строку знак находящийся на вершине, а также следующие за ним знаки с приоритетами большими или равными приоритету входного знака. После этого входной знак добавляется к вершине стека.

Открывающаяся скобка просто записывается в стек и не выталкивает ни одного знака. Но и её не может вытолкнуть ни один знак, кроме закрывающейся скобки. Появление закрывающейся скобки вызывает выталкивание всех знаков до ближайшей отрывающейся скобки включительно. Скобки взаимно уничтожаются и в выходную строку не переносятся.

После просмотра всех символов входной строки происходит выталкивание всех оставшихся в стеке символов и дописывание их к выходной строке.

Таблица приоритетов для арифметических операций приведена в таблице 6.

Таблица 6 – приоритеты для арифметических операций

Операция	Приоритет
(	0
)	1
+ –	2
* /	3

Читаем обратную польскую запись слева направо. При чтении операнда – он записывается в стек; при чтении операции – извлекаем из стека два верхних операнда, формируем тройку (операция, операнды), записываем её в матрицу; в стек записываем элемент матрицы, в котором будет результат выполнения операции.

4. Тестирование

Целью проведения тестирования является подтверждение реализации требуемой функциональной системы. Случаем, когда тестирование прошло успешно является совпадение с ожидаемым результатом.

При прохождении всех элементов лексическим анализатором и при отсутствии ошибок, на экран выводится список лексем.

При обнаружении ошибки анализатором, анализ завершает свою работу, и выводит сообщение на экран, содержащее текст ошибки.

В результате проверки правильности работы программы были получены следующие результаты:

1) Проверка лексического анализатора:

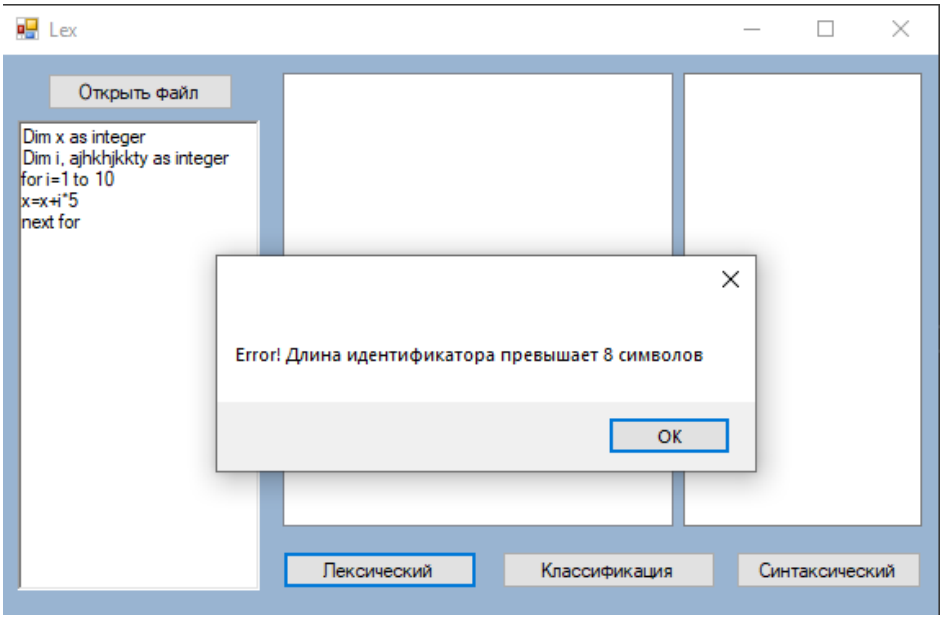

Рисунок 1 – Обнаружение ошибки ввода идентификатора лексическим анализатором

Данная ошибка обусловлена тем, что пользователь ввёл имя идентификатора длиной более 8 символов. Ошибка генерируется в методе public void Parse() класса public class Analys. Реакция программы на обнаружение ошибки представлена на рисунке 1.

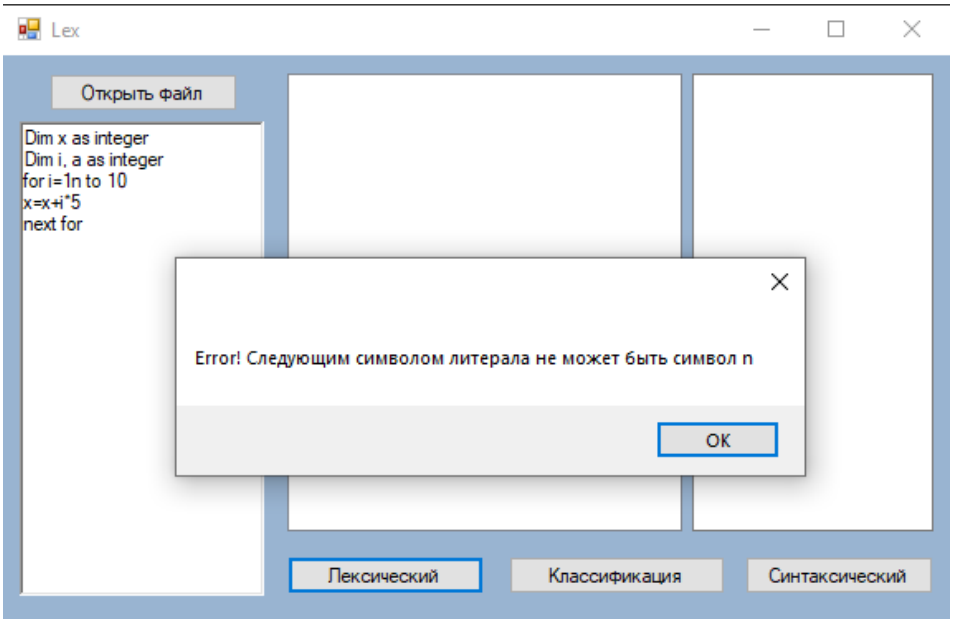

Рисунок 2 – Обнаружение ошибки ввода литерала лексическим анализатором

Данная ошибка обусловлена тем, что пользователь ввёл литерал, вторым символом которого не являлась цифра. Ошибка генерируется в методе public void Parse() класса public class Analys. Реакция программы на обнаружение ошибки представлена на рисунке 2.

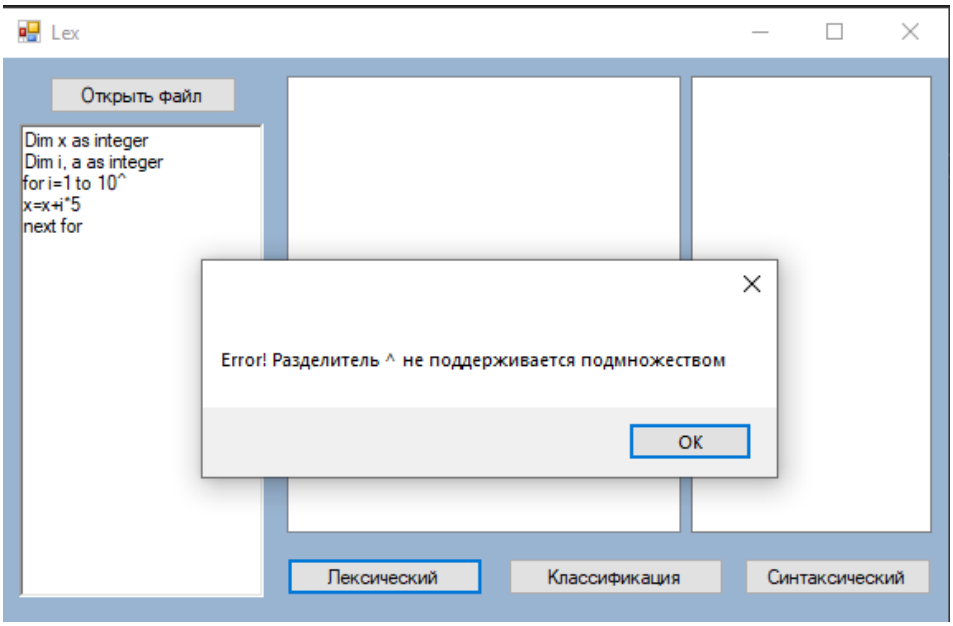

Рисунок 3 – Обнаружение ошибки ввода разделителя, неподдерживаемого подмножеством, лексическим анализатором

Данная ошибка обусловлена тем, что пользователь ввёл разделитель,

который не поддерживается подмножеством. Ошибка генерируется в методе public void Parse() класса public class Analys. Реакция программы на обнаружение ошибки представлена на рисунке 3.

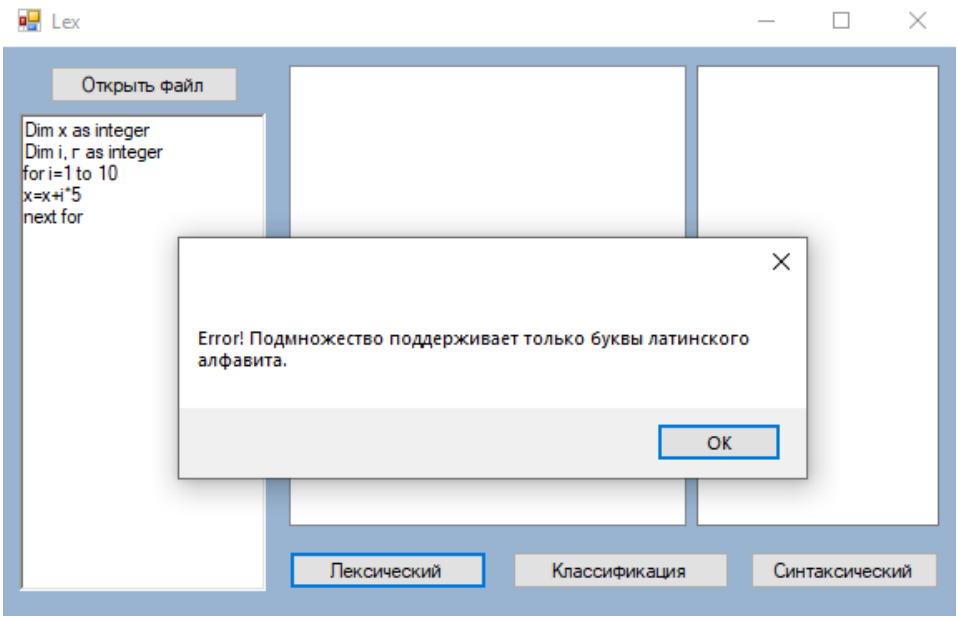

Рисунок 4 – Обнаружение ошибки языка, неподдерживаемого подмножеством, лексическим анализатором

Данная ошибка обусловлена тем, что пользователь ввёл букву из алфавита, который не поддерживается подмножеством. Ошибка генерируется в методе public void Parse() класса public class Analys. Реакция программы на обнаружение ошибки представлена на рисунке 4.

2) Проверка синтаксического анализатора:

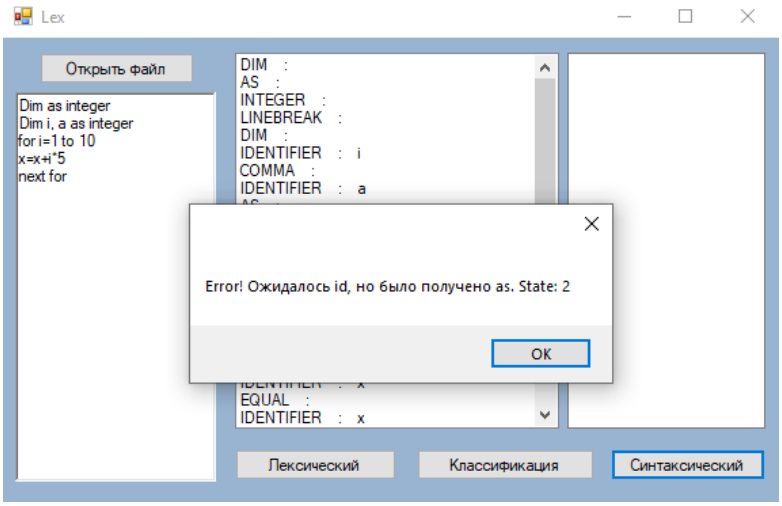

Рисунок 5 – Обнаружение ошибки в правиле <объяв.> восходящим анализатором

Данная ошибка возникает в следующем правиле грамматики: <объяв.>::=Dim <спис_перем> as <тип> \n.

После лексемы «Dim» должна быть лексема «id», соответствующая правилу <спис.идент>::= id | id, <спис.идент.>. Данная ошибка вызывается из текущего состояния. Метод обрабатывающий состояние 2 - void State2(), класс – public class LR. Реакция программы на обнаружение ошибки представлена на рисунке 5.

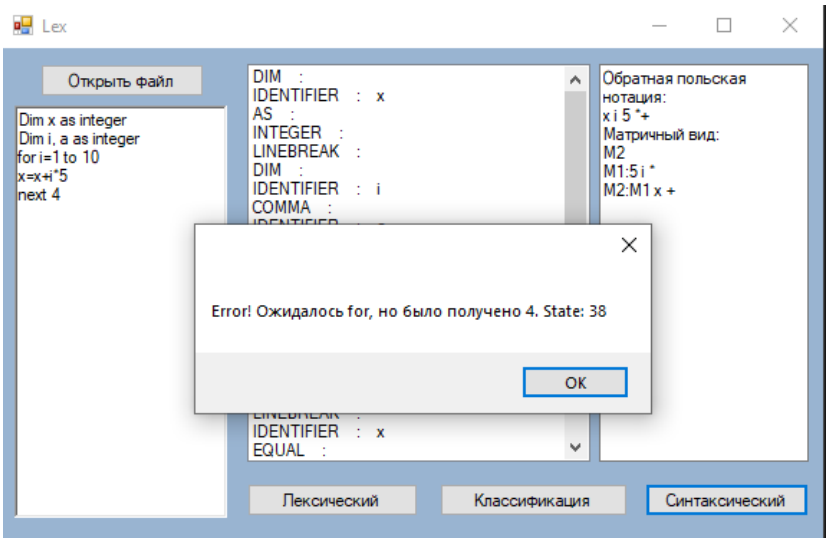

Рисунок 6 – Обнаружение ошибки в правиле <условн.> восходящим анализатором

Данная ошибка возникает в следующем правиле грамматики:

<условн.>::= for <слож.операнд> to <операнд> \n <спис.опер.> next for

После нетерминала <спис.опер> должна быть лексема «next», Соответствующий тому же правилу грамматики. Данная ошибка вызывается из текущего состояния. Метод обрабатывающий состояние 38 - void State38(), класс - public class LR. Реакция программы на обнаружение ошибки представлена на рисунке 6.

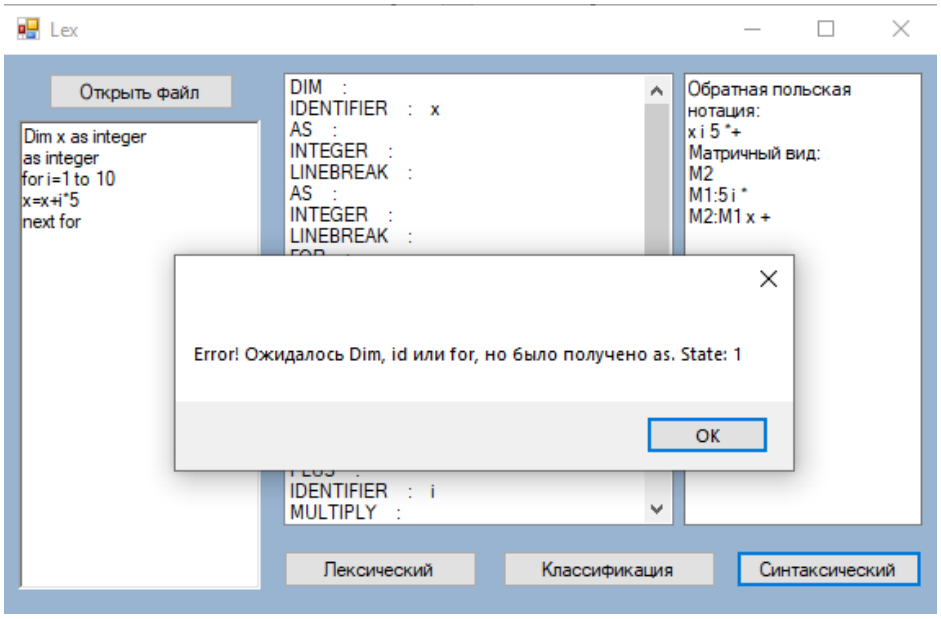

Рисунок 7 – Обнаружение ошибки в правиле <опер> восходящим анализатором

Данная ошибка возникает в следующем правиле грамматики:

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

После лексемы «\n» должна быть лексема «Dim», «id» или «for», соответствующий правилу <опер>::= <присвоен.> | <условн.> | <иниц>. Данная ошибка вызывается из текущего состояния. Метод обрабатывающий состояние 1 - void State1(), класс - public class LR. Реакция программы на обнаружение ошибки представлена на рисунке 7.

3) Проверка разбора сложных арифметических выражений:

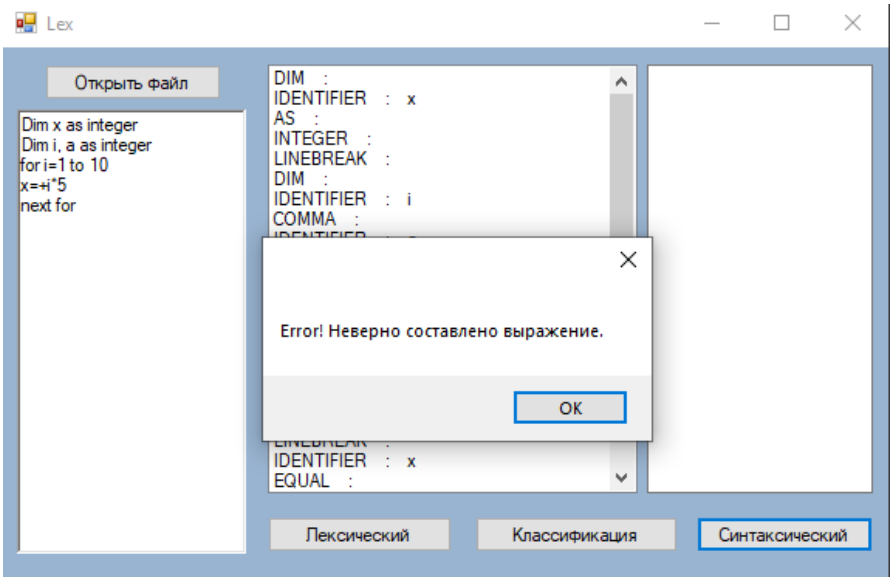

Рисунок 8 – Обнаружение ошибки в сложном арифметическом выражении алгоритмом Дейкстры

Данная ошибка обусловлена тем, что пользователь ввёл ошибочное арифметическое выражение. Данная ошибка генерируется в методе private void Decstra(), класс - public class Expression. Реакция программы на обнаружение ошибки представлена на рисунке 8.

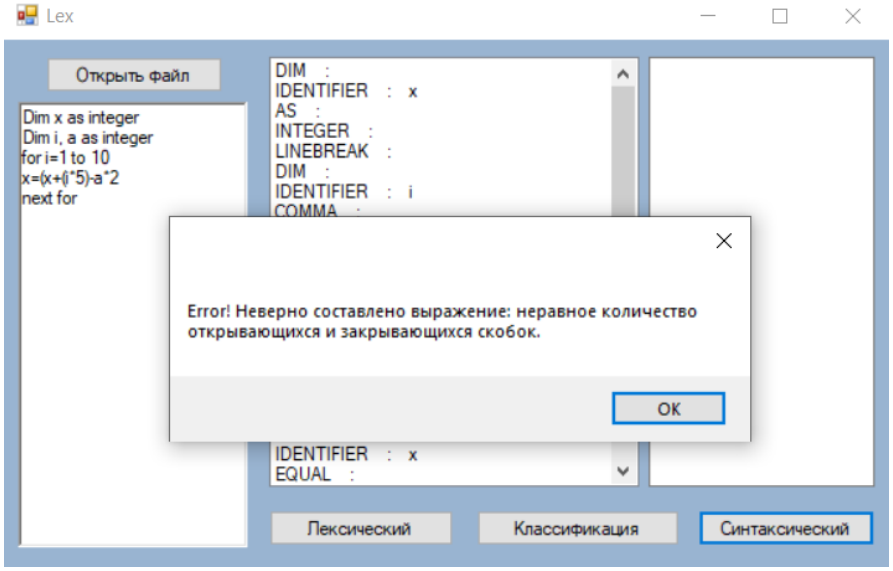

Рисунок 9 – Обнаружение ошибки в сложном арифметическом выражении при работе со скобками

Данная ошибка обусловлена тем, что пользователь ввёл неравное количество открывающихся или закрывающихся скобок в выражении. Данная ошибка генерируется в методе private void StartOPN(), класс - public class Expression. Реакция программы на обнаружение ошибки представлена на рисунке 9.

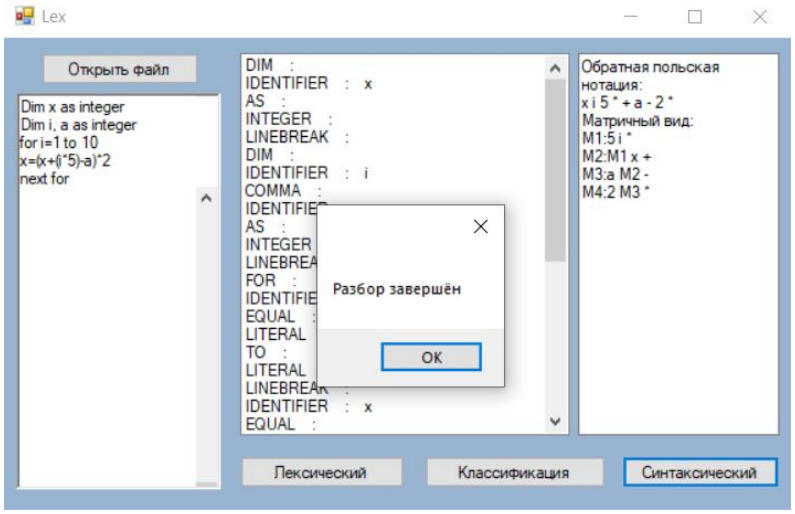

Рисунок 10 – Результат успешного завершения программы со скобками

Реакция программы на успешное завершение разбора входной последовательности представлена на рисунке 11.

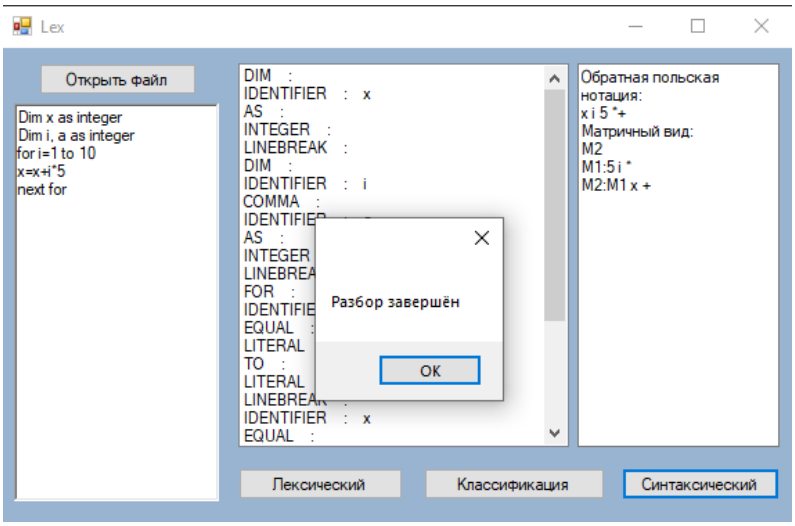

Рисунок 11 – Результат успешного завершения программы

Результаты, полученные в ходе тестирования разработанного программного продукта, позволяют сделать вывод о том, что разработанная программа соответствует требованиям технического задания.

5. Руководство пользователя

Данное приложение создано для выполнения лексического, синтаксического анализа, а также разбора сложных логических выражений подмножества языка Visual Basic.

В приложении реализованы:

- лексический разбор;
- классификация лексем, полученных в результате лексического разбора;
- синтаксический разбор на основе LR(k)-грамматик;
- разбор сложных арифметических выражений методом Дейкстра;

При открытии приложения пользователь может видеть: кнопку «Открыть файл», с помощью которой может загрузить содержимое текстового файла на форму. Присутствует возможность ввести текст для анализа или редактировать загруженный текст вручную. Кнопка «Лексический», проводит лексический анализ текста, находящегося на форме. Кнопка «Классификация» выводит на экран типизированный список лексем, полученный в результате лексического анализа. В случае если лексема не поддерживается подмножеством, выводится сообщение об ошибке. При нажатии кнопки «Синтаксический», проводится восходящий анализ введенного кода, с использованием полученных при выполнении классификации лексем. В случае, если список лексем соответствует грамматике выводится сообщение «Разбор завершён», иначе выведется сообщение с ошибкой. Также, если в анализируемом фрагменте присутствует сложное арифметическое выражение, то результат его разбора будет выведен на форму в виде текста в форме.

6. Руководство программиста

Входными данными для транслятора является текст программы, полученный от пользователя, с синтаксисом подмножества языка программирования VB. Выходными данными транслятора являются таблица лексического разбора программы, список ключевых слов, разделителей, идентификаторов и литералов, используемых в программе после выполнения лексического анализа. В случае обнаружения ошибки в одном из анализаторов программа может вывести сообщение об обнаружении ошибки синтаксическим анализатором. Ниже в таблице 9 представлены основные методы программы и их назначения:

Таблица 9 – Классы программы и их назначения

Метод	Назначение
public class Analys	Класс для лексического анализа
public void Parse()	Метод разбора исходного фрагмента текста на категории идентификатор, литерал и разделитель
public struct Lex	Структура для хранения лексемы
public Lex (string L, string Type)	Конструктор структуры Lex, где string L – строка записываемая в поле Lexema, string Type – строка записываемая в поле Type
public class Token	Класс для токенизации
public TokenType Type	Поле для хранения типа токена
private static TokenType[] Delimiters	Массив, содержащий типы разделителей подмножества
public static Dictionary<string, TokenType> SpecialWords	Словарь ключевых слов подмножества
public static Dictionary<char, TokenType> SpecialSymbols	Словарь разделителей подмножества
public enum TokenType	Перечисление возможных типов лексем
public class LR	Класс для восходящего синтаксического анализа
public LR(List<Token> vvodtoken)	Конструктор класса LR, где vvodtoken List<Token> список токенов для анализа

private Token GetLexeme(int nextLex)	Метод, возвращающий токен из списка токенов, где nextLex int целочисленное значение, содержащее номер анализируемого токена из списка токенов. Возвращаемое значение Token - Токен из списка токенов
private void Shift()	Метод сдвига очередного символа из входной строки в стек с переходом к следующему
private void GoToState(int state)	Метод перевода анализатора в другое состояние, где state int целочисленное значение, содержащее номер текущего состояния анализатора.
private void Reduce(int num, string neterm)	Свёртка, метод замены, хранимой в стеке разбора правой части правила на левую, где int num целочисленное значение, содержащее число удаляемых из стеков символов, string neterm - строка, содержащая название нетерминала, отправляемого в стек разбора.
private void Expr()	Метод разбора сложных логических выражений
void State0() … void State51()	Методы состояний, действия которых соответствует решающей таблице, приведённой в таблице 5. В случае ошибки в данных методах вызывается Exception, где пишутся ожидаемые для разбора символы или правила.
public void Start()	Метод запуска восходящего анализатора
public class Expression	Класс для разбора сложных логических выражений
Dictionary<string, int> priority	Список приоритетов для операций
public void TakeToken(Token token)	Получение из анализатора логического выражения, где Token token объект класса Token
public void StartOPN()	Метод запуска разбора сложного выражения

private void HighPriority(string operation)	Вспомогательная функция для метода Дейкстры, где string operation строка, содержащая текущую операцию
private void Decstra()	Метод Дейкстры, где мы выталкиваем операнды, а следом за ними знаки
public void PolishNotation()	Метод выполняющий преобразование обратную польскую нотацию в матричный вид
public partial class Form1 : Form	Класс для создания приложения winforms
public void Conclusion1()	Метод, выполняющий очистку текста
public void Conclusion(string EXPR)	Метод записи обратной польской нотации и матричного вида, где string EXPR – строка, содержащая информацию вывода обратной польской нотации и матричного вида
private void button1_Click (object sender, EventArgs e)	Метод вызова лексического анализатора
private void button2_Click(object sender, EventArgs e)	Метод вызова классификации лексем
private void button3_Click(object sender, EventArgs e)	Метод для импорта кода из .txt файла
private void button4_Click(object sender, EventArgs e)	Метод вызова разбора на основе LR грамматики

Заключение

В результате работы создан транслятор программы на подмножестве языка Visual Basic. Данная программа может выполнять лексический, синтаксический анализы, а также разбор сложного логического выражения. Для синтаксического разбора был использован метод LR(k)-грамматик. Для разбора сложного арифметического выражения использован метод Дейкстры.

В процессе работы была составлена грамматика подмножества языка Visual Basic, реализованы методы лексического и синтаксического разбора полученного кода, а также выполнено тестирование программы.

Для более подробного ознакомления с приложением можно воспользоваться ссылкой на репозиторий данного проекта: https://github.com/focs1t/PIn-121-Maresev-Sergey.git

Подводя итоги, можно считать, что разработанный транслятор соответствует требованиям технического задания.

Список используемой 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.

4. Миронов, С. В. Формальные языки и грамматики : учебное пособие для студентов факультета компьютерных наук и информационных технологий / С. В. Миронов. — Саратов : Издательство Саратовского университета, 2019. — 80 с.