hМинистерство 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)

Мочалин Н.А.

(фамилия, инициалы)

(подпись) (дата)

Муром 2023

Главная цель этой курсовой работы - это разработать транслятор подмножества языка программирования Vizual Basic. Разработка велась на языке программирования C# с использованием фреймворка .NET в Microsoft Vizual Studio 2022.

The main goal of this course work is to develop a translator for a subset of the Vizual Basic programming language. The development was carried out in the C# programming language using a framework.NET in Microsoft Visual Studio 2022.

Содержание

Введение5
1. Анализ технического задания 6
2. Описание грамматики языка 7
3. Разработка архитектуры системы и алгоритмов 12
4. Тестирование 17
5. Руководство пользователя 22
6. Руководство программиста: 24
Заключение 27
Список используемой литературы 28
Приложения 30

Введение

В современном мире, где информационные технологии занимают все большую роль, разработка трансляторов является одной из самых важных задач. Трансляторы позволяют переводить программы на одном языке программирования в программы на другом языке, что упрощает их разработку и совместимость.

В данной курсовой работе мы рассмотрим основные принципы разработки транслятора для языка Visual Basic. Мы изучим его особенности и применения, а также рассмотрим примеры использования в реальных проектах.

Исходными данными для транслятора служит текст входной программы, т.е некая последовательность символов входного языка программирования, удовлетворяющая синтаксическим требованиям.

Идея транслятора легла в основу создания многих языков программирования, например Visual Basic - процедурного языка с элементами компонентной и структурной парадигмой программирования.

1. Анализ технического задания

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

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

1) развернутая диагностика ошибок.

2) реализация класса транслятора.

3) синтаксический разбор – на основе LL(k)-грамматик.

4) разбор выражений, выполненный методом Дейкстры.

В языке поддерживаются:

1) у идентификатора 8 символов значащие.

2) не менее трех директив описания переменных.

3) простой арифметический оператор.

4) сложное логическое выражение;

5) оператор цикла DO WHILE … LOOP

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.

Программа реализуется на языке C# с использованием фреймворка .NET.

Программа представляет собой приложение Windows Forms.

Ввод кода VB может производится как с клавиатуры напрямую, так и с помощью открытия текстового файла содержащий код программы.

2. Описание грамматики языка

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. 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.

Оператор цикла DO WHILE ... LOOP используется если нужно выполнять цикл пока условие верно.

Грамматика языка:

G(N, T, P, <программа>)

T = { Dim, as, loop,+, -, /, *, =>, <, >=, <=, <> ,=, (,), id, lit, do, while, expr}

N={<программа>,<спис_опис>,<опис>,<тип>, <спис_перем>, <опер> , <условн.>, <блок.опер>,<присв.>, <знак>,<операнд>}

P =

{

<программа>::=Dim <спис_опис> < спис_опер>

<спис_опис> ::=<опис> | <спис_опис> <опис> <опис>::=<спис_перем> as <тип> \n

<спис_перем> ::= id | <спис_перем>, id

<тип>::= integer | long | double <спис_опер>::= <опер> | <спис_опер>\n<опер>

<опер>::=<условн.> | <присв.>

<условн.> ::= do while expr \n <блок опер.> loop \n

<блок опер.> ::= <опер>\n|<спис_опер>\n

<присв.>::= id= <операнд> <знак> <операнд> \n | id= <операнд> \n

<знак>::= + | - | * | /

<операнд>::=id | lit

}

Таблица 1 - Пример формирования цепочки вывода

Анализируемый фрагмент программы	Полученный синтаксический вывод
Dim a as integer b = 1 do while (a < 10 or b > c) b = b + a loop	Dim id as integer id = lit do while (id < lit or id > id) id = id + id loop

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

Избавимся от левой рекурсии и левой факторизации:

1) <спис_опис> ::=<опис> | <спис_опис>\n <опис>

<спис_опис> ::=<опис> <Y>

<доп_опис> ::= \n <опис> <Y>

<Y>::= Ɛ | <доп_опис>

2) <спис_перем> ::= id |<спис_перем>, id

<спис_перем>::= id <X>

<доп_перем>::= , id <X>

<X>::= ε | <доп_перем>

3) <спис_опер>::= <опер>\n | <спис_опер> \n <опер>\n

<спис_опер> ::= <опер>\n <Z>

<доп_опер>::= <опер> <Z>

<Z>::= ε | <доп_опер>

4) <присв>::= id= <операнд> <знак> <операнд> | id= <операнд>

<присв>::= id= <операнд><U>

<U> ::= <знак> <операнд> | ε

Полученная грамматика:

G(N, T, P, <программа>)

T = { Dim, as, loop,+, -, /, *, =>, <, >=, <=, <> ,=, (,), id, lit, do, while, expr}

N={<программа>,<спис_опис>,<опис>,<тип>, <спис_перем>, <опер> ,

<условн.>, <блок.опер>,<присв.>, <знак>,<операнд>}

P =

{

<программа>::=Dim <спис_опис> < спис_опер>

<спис_опис> ::=<опис> <Y>

<доп_опис> ::= \n <опис> <Y>

<Y>::= Ɛ | <доп_опис>

<опис> ::=<спис_перем> as <тип> \n

<спис_перем>::= id <X>

<доп_перем>::= , id <X>

<X>::= ε | <доп_перем>

<тип>::= integer | long | double

<спис_опер> ::= <опер>\n <Z>

<доп_опер>::= <опер> <Z>

<Z>::= ε | <доп_опер>

<опер>::=<условн.> | <присв.>

<условн.> ::= do while expr \n <блок опер.> loop \n

<блок опер.> ::= <опер>\n|<спис_опер>\n

<присв>::= id= <операнд><U>

<U> ::= <знак> <операнд> | ε

<знак> ::= + | - | * | /

<операнд>::=id | lit

}

Из таблицы 2 можно сделать вывод что преобразования грамматика принадлежит к LL(1):

Таблица 2 – решающая таблица нисходящего анализа

Правила грамматики	$First_1 \cup Follow_1$
<программа>::=Dim <спис_опис> \n <спис_опер>\n	Dim
<спис_опис> ::=<опис> <Y>	id
<доп_опис> ::= \n <опис> <Y>	\n
<Y>::= Ɛ <Y>::=<доп_опис>	$ \n
<опис> ::=<спис_перем> as <тип> \n	id
<спис_перем>::= id <X>	id
<доп_перем>::= , id <X>	,
<X>::= ε <X>::= <доп_перем>	$,
<тип>::= integer <тип>::= long <тип>::= double	integer long double
<спис_опер> ::= <опер>\n <Z>	do, id
<доп_опер>::= <опер> <Z>	do, id
<Z>::= ε <Z>::= <доп_опер>	$
<опер>::= <условн.> <опер>::= <присв.>	do id
<условн.> ::= do while expr \n <блок опер.> loop \n	do
<блок опер.> ::= <опер>\n|<спис_опер>\n	do, id
<присв>::= id = <операнд><U>	id
<U> ::= <знак> <операнд> \n | \n <U> ::= ε	+,-,*,/ $
<знак> ::= + <знак> ::= - <знак> ::= * <знак> ::= /	+ - * /
<операнд>::= id <операнд>::= lit	id lit

3. Разработка архитектуры системы и алгоритмов

Во время 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 проверкой 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». 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 на представлен в таблицах 3 и 4:

Таблица 3-Пример работы сканера

Dim	Идентификатор
\n	Конец строки
=	Разделитель
3	Литерал
do	Идентификатор
+	Разделитель

Таблица 4 -Пример работы лексического анализатора

Dim	IDENTIFIER
\n	ENTER
=	EQUAL
3	NUMBER
do	IDENTIFIER
+	PLUS

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

В задачу синтаксического анализа входит: найти и выделить основные синтаксические конструкции в тексте входной программы, установить тип и проверить правильность каждой синтаксической конструкции, наконец, представить синтаксические конструкции в виде, удобном для дальнейшей генерации текста результирующей программы.

Распознаватель дает ответ на вопрос о том, принадлежит или нет цепочка входных символов заданному языку – это основная задача синтаксического анализатора. Кроме этого синтаксический анализатор должен иметь некий

выходной язык, с помощью которого он передает следующим фазам компиляции всю информацию о найденных и разобранных синтаксических структурах.

Синтаксический анализатор воспринимает выход лексического анализатора и разбирает его в соответствии с грамматикой входного языка. Однако в грамматике входного языка программирования обычно не уточняется, какие конструкции следует считать лексемами. Примерами конструкций, которые обычно распознаются во время лексического анализа, служат ключевые слова, константы и идентификаторы. Но эти же конструкции могут распознаваться и синтаксическим анализатором. На практике не существует жесткого правила, определяющего, какие конструкции должны распознаваться на лексическом уровне, а какие надо оставлять синтаксическому анализатору. Обычно это определяет разработчик компилятора исходя из технологических аспектов программирования, а также синтаксиса и семантики входного языка.

Используя токены лексический анализ производит токенизацию, т.е процесс классификации разделов строки входных символов. Большая часть методов анализа введенных данных принадлежат либо нисходящим алгоритмам, либо восходящим. Нисходящие анализаторы строят вывод, начиная от аксиомы грамматики и заканчивая цепочкой терминальных символов. Нисходящий анализ связан с LL – грамматикой, которая обладает следующими особенностями:

Она может быть проанализирована без возвратов.

Первая буква L означает, что входная цепочка проверяется слева направо.

Вторая буква L означает, что строится левый вывод цепочки.

В данной курсовой работе осуществлен нисходящий анализатор LL(1). В Таблице 2 представлена решающая таблица нисходящего анализатора:

Алгоритм работы нисходящего анализатора:

На вход поступает список лексем, полученных при работе сканера. Далее программа, основываясь на полученной грамматике, проводит проверку полученного списка лексем на соответствие с грамматикой. В случае, если полученные лексемы совпадают со списком грамматик, выводится сообщение, что ошибок не обнаружено. В случае если полученные лексемы не совпадают со списком грамматик, выводится сообщение, в котором указывается ожидаемый и полученный символы.

Алгоритм работы разбора сложных логических выражений(Метод Дейкстры):

1. Входная строка просматривается слева направо, для каждого символа:

1.1. если символ является операндом, то он добавляется к выходной строке;

1.2. если символ является открывающейся скобкой, она помещается в стек;

1.3. если символ является закрывающейся скобкой, выталкиваются элементы из стека в выходную строку до тех пор, пока на вершине стека не окажется открывающаяся скобка.

При этом открывающаяся скобка удаляется из стека, но в выходную строку не добавляется.

Если стек закончился раньше, чем встретилась открывающая скобка - в выражении несогласованны скобки.

1.4. если символ операция, то:

- если приоритет операции равен нулю или больше приоритета операции, находящейся на вершине стека или стек пуст, то входная операция записывается в стек,
- иначе из стека выталкивается и переписывается в выходную строку операция находящаяся на вершине, а также следующие за ней операции с приоритетами большими или равными приоритету входной операции. После этого входная операция добавляется к вершине стека.

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

Далее осуществляется перевод обратной Польской нотации в матричный вид по правилам:

1. ОПН просматривается слева направо;
2. При чтении операнда – он записывается в стек операндов;
3. При чтении операции:
   - из стека извлекается два верхних операнда;
   - формируется тройка (операция, операнды) и записывается в матрицу операций;
   - в стек записывается обозначение строки матрицы с полученной операцией.

4. Тестирование

Неотъемлемая часть разработки это тестирование, оно нужно чтобы проверить программу на корректность работы.

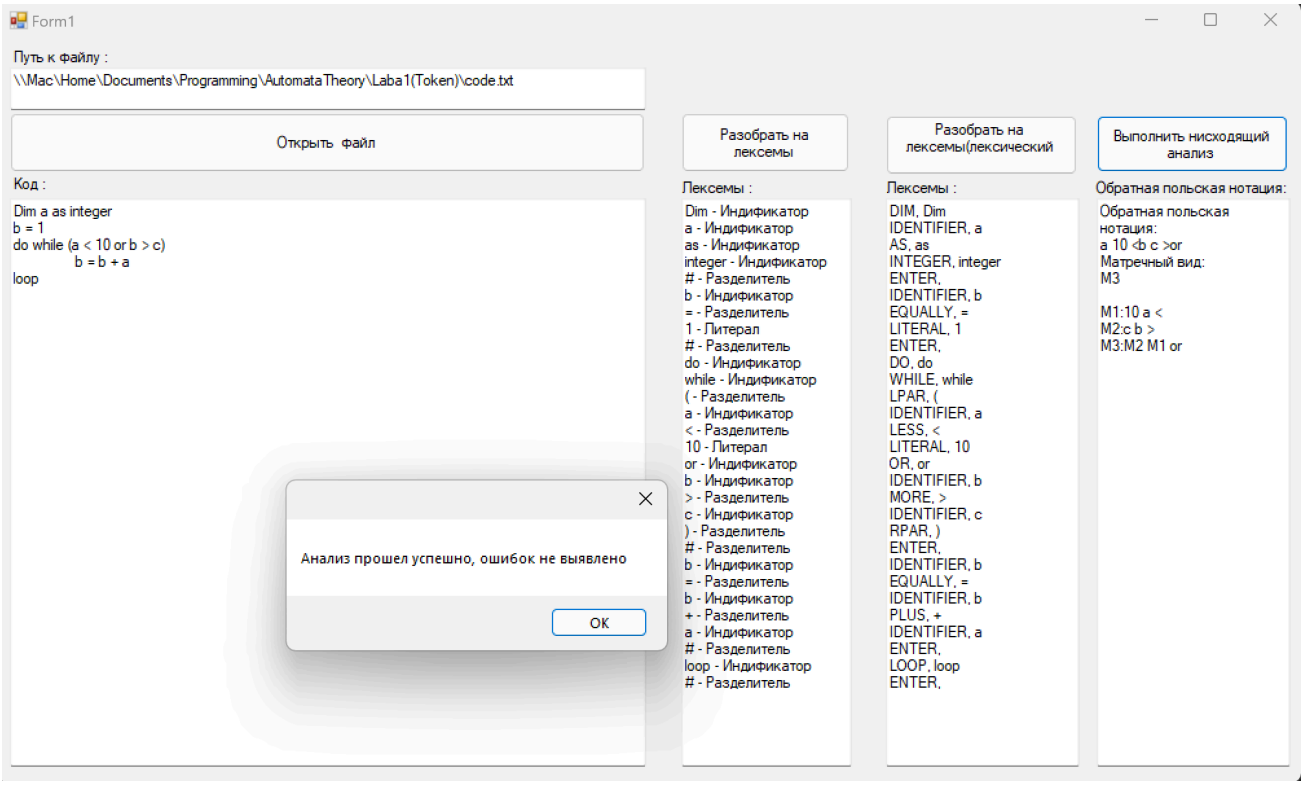

Рисунок 1 - тестирование корректного VB кода

На рисунке 1 приведён пример корректного кода VB. В ходе его анализа не обнаружено ошибок, так как их нет.

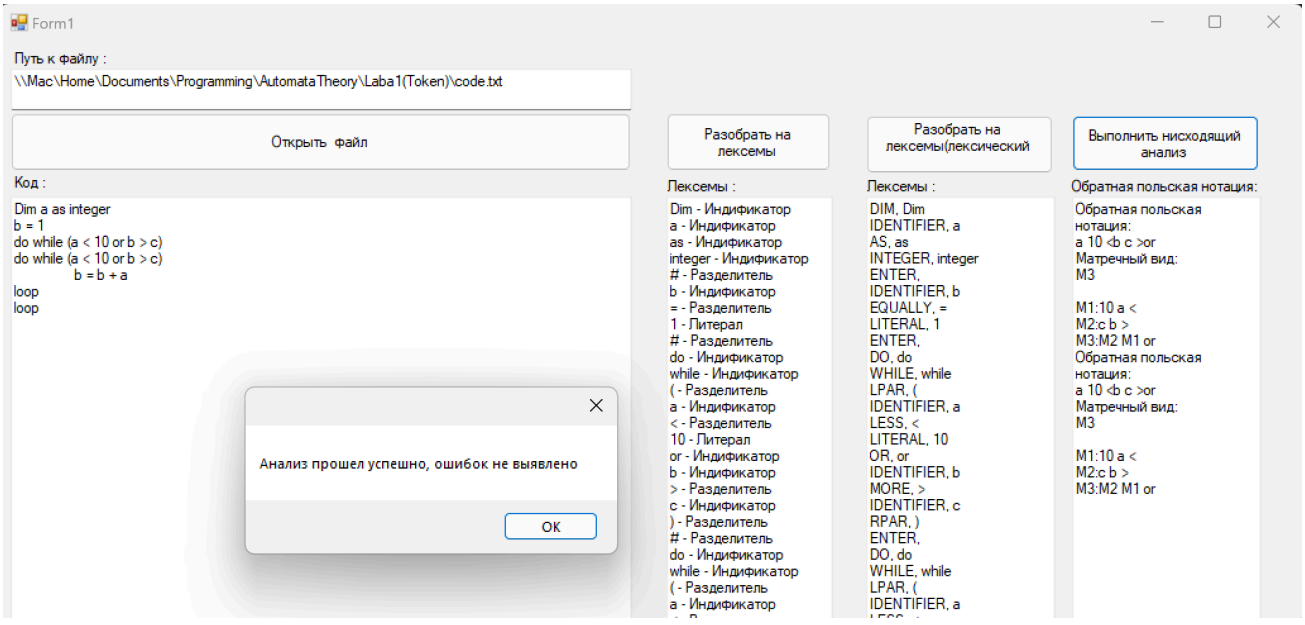

Рисунок 2 - тестирование кода VB с вложенным циклом

На рисунке 2 приведён пример корректного кода VB с вложенным циклом. В ходе его анализа не обнаружено ошибок, так как их нет.

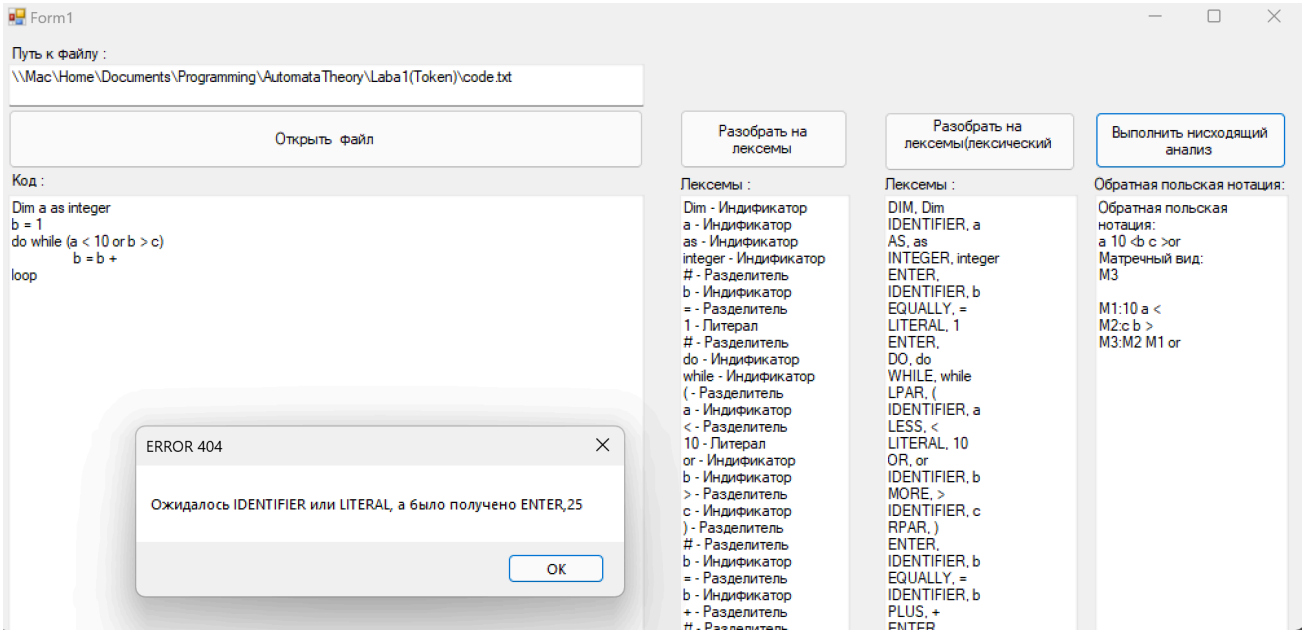

Рисунок 3 - тестирование некорректного кода VB

На рисунке 3 приведён пример некорректного кода VB в 4 строчке допущена ошибка в присвоение, если в присвоение есть знак значит, после знака арифметической операции должен идти идентификатор или литерал.

В методе Operand описана данная ошибка

<операнд>::= id <операнд>::= lit	id lit

Правило которому соответствует ошибка представлено ниже

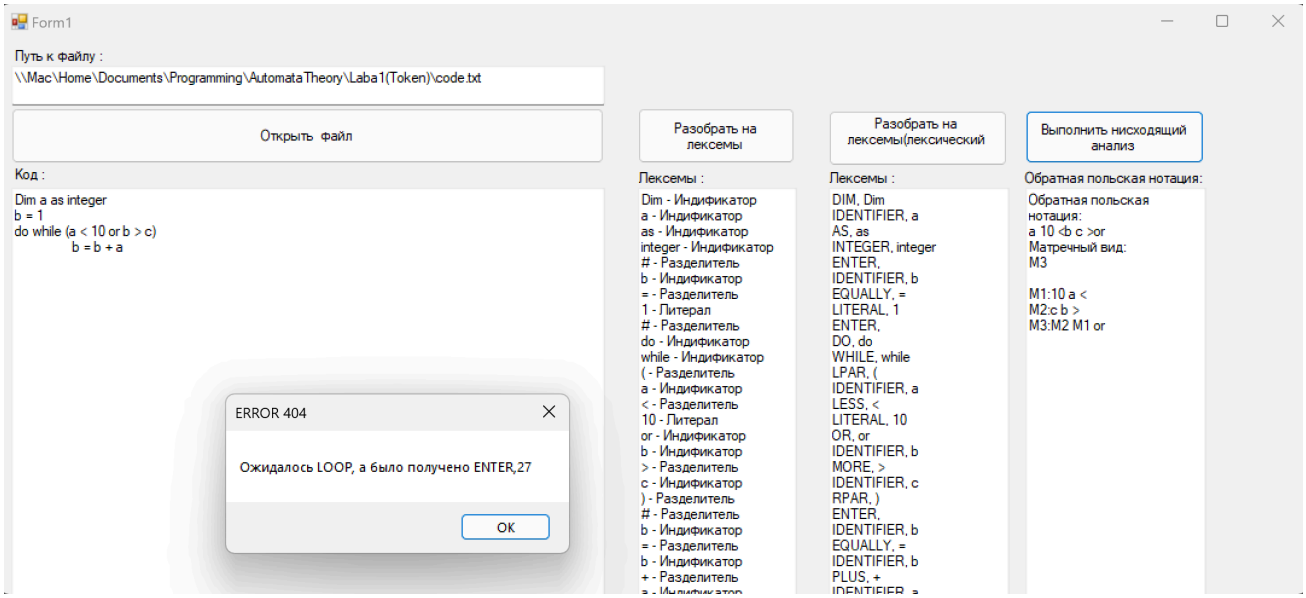

Рисунок 4 - тестирование кода VB с некорректным кодом

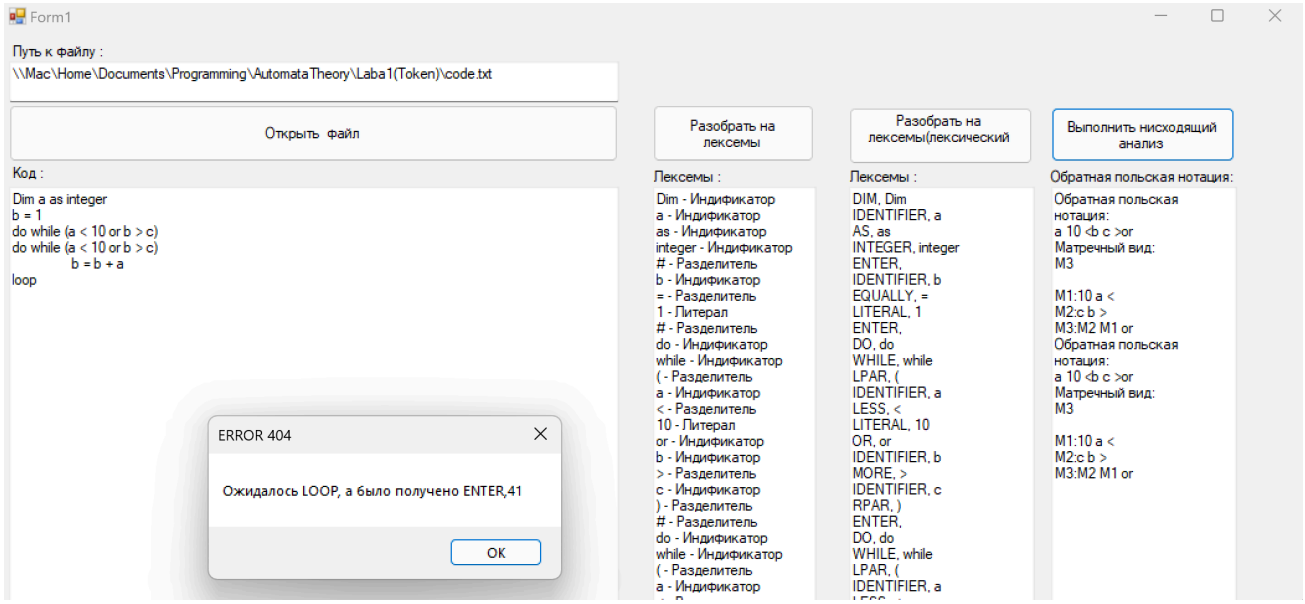

Рисунок 5 - тестирование кода VB с некорректным кодом

На рисунке 4,5 приведены примеры некорректного кода VB. В данных примерах имеется ключевые слова, открывающие цикл(do while), но отсутствует закрывающее ключевое слово(loop).

<условн.> ::= do while expr \n <блок опер.> loop \n	do

Правило которому соответствует ошибка представлено ниже

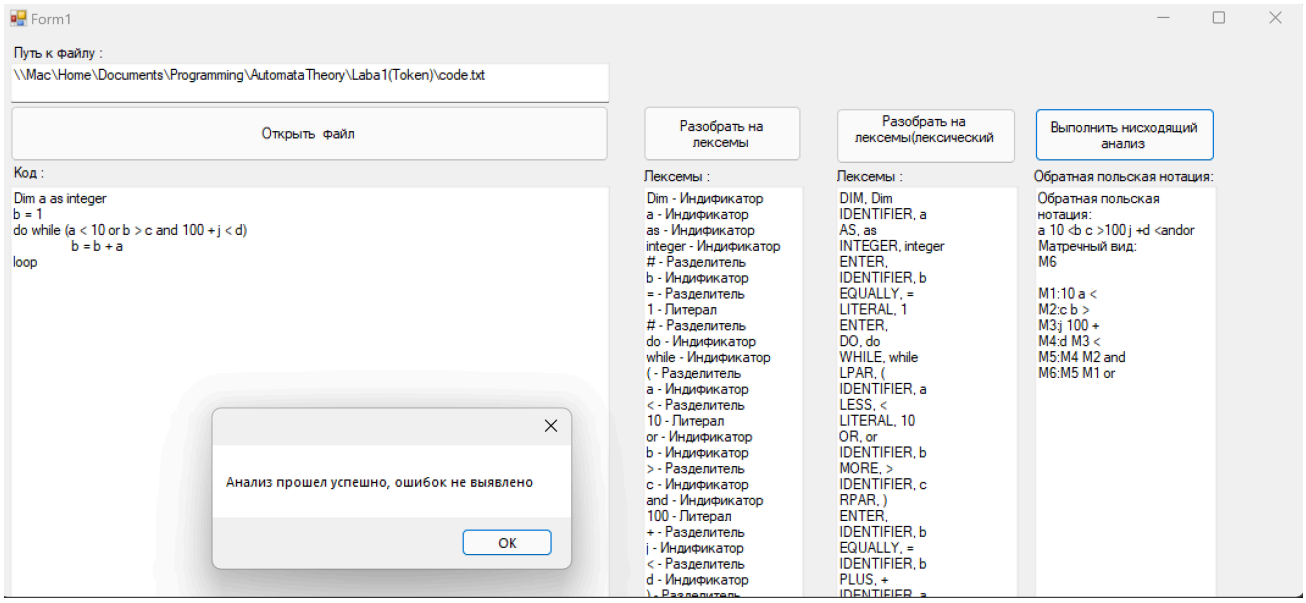

Рисунок 6 - тестирование корректного VB кода

На рисунке 6 приведён пример кода VB с сложным логическим выражение в котором присутствует арифметическая операция. Ошибок не выявлено, так как в логических выражения допускаются арифметические операции

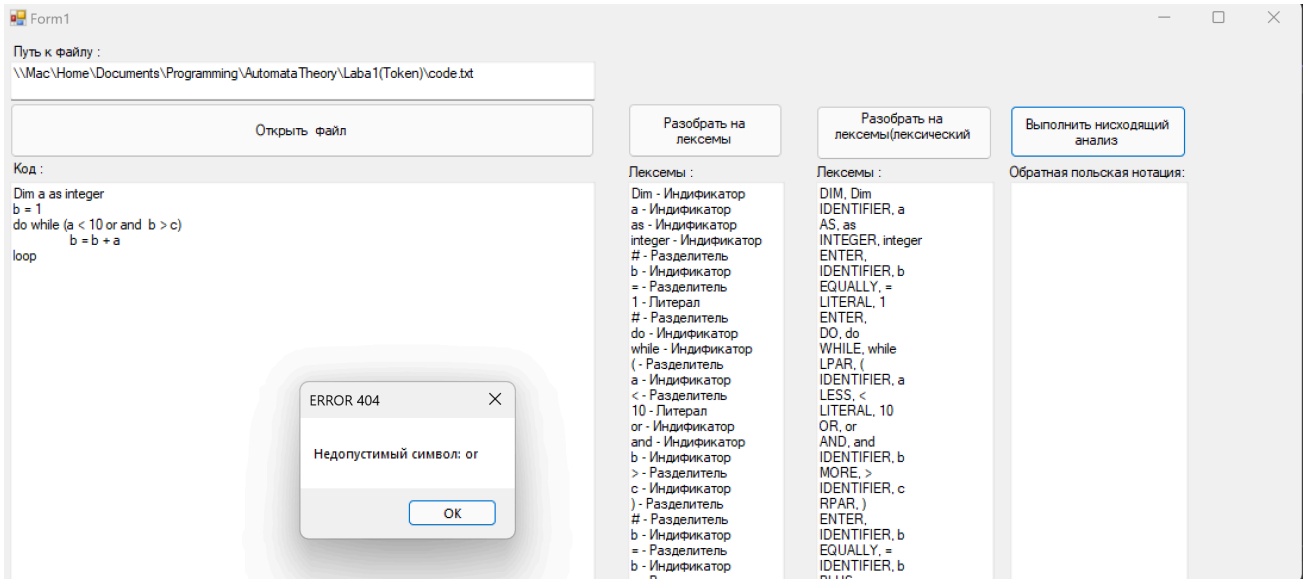

Рисунок 7 - тестирование кода VB с некорректным кодом

На рисунке 7 приведён пример кода VB с некорректным сложным логическим выражением.

Перед and не может стоять or, так как перед and ожидается сравнение.

Данная ошибка описана в методе ReversePolishNotationInMatrixView()

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

5. Руководство пользователя

Данное приложение создано для выполнения лексического и синтаксического анализа подмножества языка Visual Basic. Приложение имеет поддержку синтаксического разбора на основе LL(k)-грамматик, сложный арифметический оператор, оператор выбора do while … loop.

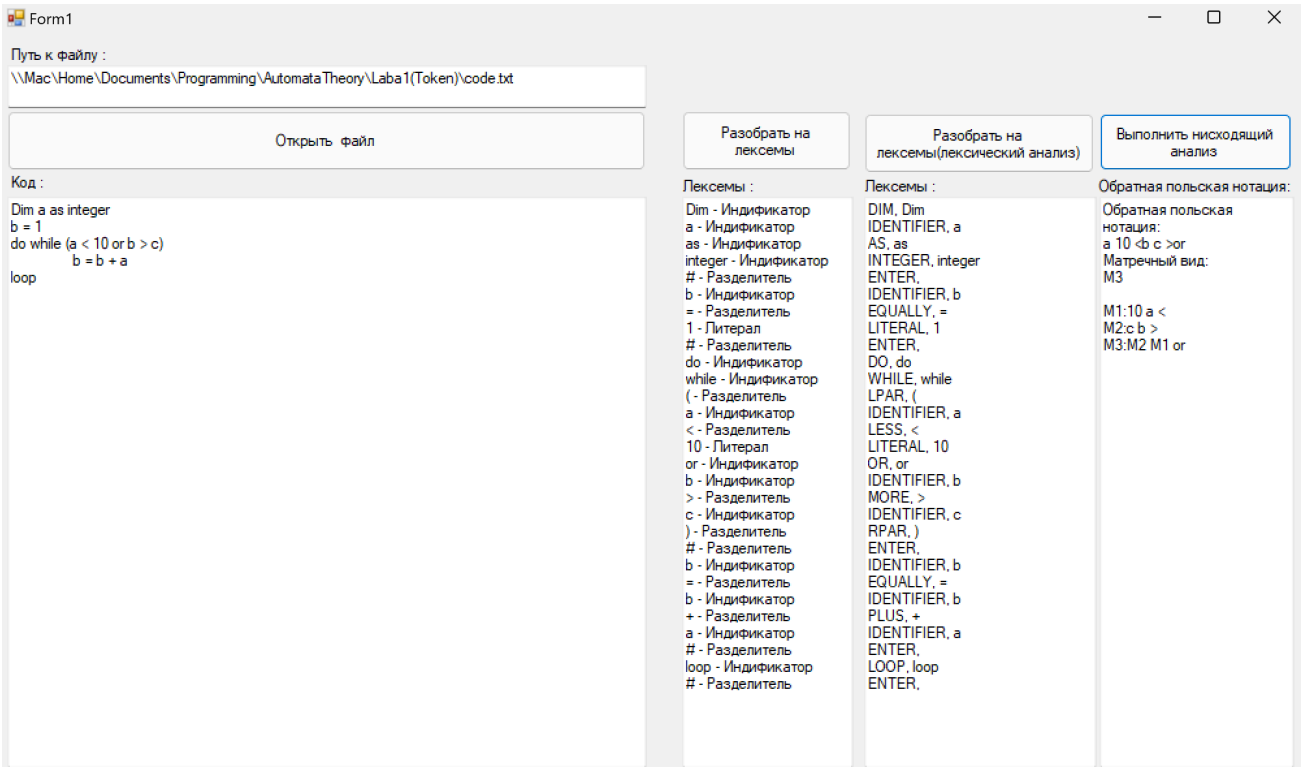

Рисунок 8 - интерфейс программы.

На рисунке 8 представлен интерес программы. Слева сверху можно видеть ячейку для ввода текста под надписью «Путь к файлу:», данная ячейка служит для ввода пути к текстовому файлу с кодом VB. Это поле стоит заполнить в первую очередь, если транслировать код из текстового файла. После заполнения следует нажать кнопку «Открыть файл», в случае если путь к текстовому файлу не верны то произойдёт ошибка. Если вы собираетесь вводить код VB в ручную то это стоит делать в ячейке для ввода текста под надписью «Код:». Также в этой ячейке можно исправлять код полученный из текстового документа. После того как код VB введён, нажимаем на кнопку «Разобрать на лексемы». После этого в ячейке под этой кнопкой будут показаны все лексемы поделённые на идентификаторы, литералы и разделители.

Далее нажимаем на кнопку «Разбор на лексемы (лексический анализ)», после этого в ячейке под кнопкой будут отображены все лексемы и их классификация. После всех выше описанных действий нажимаем на кнопку «Выполнить нисходящий анализ», данная кнопка отвечает за выполнения нисходящего анализа и вывода обратной польской нотации и матричного вида сложного логического выражения в ячейку под ней. В случае если в коде VB опущенная ошибка, то программа выведет сообщение об ошибке.

6. Руководство программиста:

Выходными данными транслятора являются таблица лексического разбора программы, список ключевых слов, разделителей, идентификаторов и литералов, используемых в программе после выполнения лексического анализа.

Form1.cs

Form1.cs отвечает за интерфейс и его корректное функционирование

button1_Click отвечает за открытие файла

button2_Click разбирает на лексемы, реализуя класс Volidate.cs.

button3_Click выполняет лексический анализ, реализуя Token.cs.

button4_Click выполняет нисходящей анализ, Analyzer.cs

Volidate.cs

Volidate.cs посимвольно проверяет код на коректность введённых символов.

_separators массив символов, которые являются разделителями

Метод VolidateIsID принимает на вход символ, проверяет является ли этот символ идентификатором и передаёт на выход булевое значение, где true означает что принятый символ является идентификатором, а false - не является.

Метод VolidateIsLiteral принимает на вход символ, проверяет является ли этот символ литералом и передаёт на выход булевое значение, где true означает что принятый символ является литералом, а false - не является.

Метод VolidateIsSeparator принимает на вход символ, проверяет является ли этот символ разделителем и передаёт на выход булевое значение, где true означает что принятый символ является разделителем, а false - не является.

Token.cs

Token.cs содержит в себе словарь для записи лексем и методы которые реализуют данную запись

TokenType - это список токенов

Delimiters - это разделители

Метод IsDelimiter на вход принимает токен и проверяет на принадлежность к разделителям и возвращает true или false в зависимости от того является токен разделителем или нет

SpecialWords - является словарям для всех терминалов кроме разделителей

Метод IsSpecialWord Принимает на вход строковою переменную и возвращает true или false в зависимости от того является ли слово терминалом или нет

Analyzer.cs

Analyzer.cs содержит методы реализующие граматику LL(k)1

Методы Exception выводят определённое сообщение на экран в зависимости от того сколько передано переменных.

Функция Next осуществляет приход к следующей лексеме

Функция Begin является входной точкой в класс Analyzer.cs

Метод Expresion является обработчиком сложного логического выражения

Функция Programm реализует правило граматики «<программа>»

Функция ListOfDescriptions реализует правило граматики «<спис_опис>»

Функция AdditionalDescription реализует правило граматики «<доп_опис>»

Функция Description реализует правило граматики «<опис>»

Функция ListOfVariables реализует правило граматики «<спис_перем>»

Функция AdditionalVariables реализует правило граматики «<доп_перем>»

Функция X реализует правило граматики «X»

Функция Type реализует правило граматики «<тип>»

Функция ListOfOperators реализует правило граматики «<спис_опер>»

Функция AdditionalOperator реализует правило граматики «<доп_опер>»

Функция Z реализует правило граматики «Z»

Функция Operator реализует правило граматики «<опер>»

Функция Conditional реализует правило граматики «<условн.>»

Функция OperatorBlock реализует правило граматики «<блок опер.>»

Функция Assignment реализует правило граматики «<присв>»

Функция U реализует правило граматики «U»

Функция Sign реализует правило граматики «<знак>»

Функция Operand реализует правило граматики «<операнд>»

Expression.cs

Метод TakeToken получает на вход список токинов, которые в в последствие добавляются в список logicExpressionStack

Словарь priority содержит в себе приоритеты для каждой операции

Метод Start является входной точкой в программу и вызывает 2 метода Decstra() и ReversePolishNotationInMatrixView();

Метод Decstra выполняет метод Дейкстры и в переменную output записывают обратную Польскую нотацию

Метод PushesOutOperationsHighestPriority принимает на вход строковую форму операции, является вспомогательным методом для метода Decstra и выполняет выталкивание всех операций выше приоритета, чем входная операция из стека stackOfOperations в выходную строку output

Метод ReversePolishNotationInMatrixView переводит Обратную Польскую нотацию в матречный вид и выводит на форму

Заключение

В ходе курсовой работы были выполнены все поставленные цели, как следствие разработан транслятор для подмножества языка Visual Basic. Данный транслятор может разбирать и переводить в матричный вид сложные логические выражения с помощью метода Дейкстры. Также он проводит синтаксический разбор - на основе LL(k)-грамматик.

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

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. — ISBN 987-5-7433-
2968-7. — Текст : электронный // Электронно-библиотечная система IPR BOOKS :
[сайт]. — URL: https://www.iprbookshop.ru/76519.html (дата обращения: 15.04.2021). — Режим доступа: для авторизир. пользователей. - DOI: https://doi.org/10.23682/76519 - https://www.iprbookshop.ru/76519.html

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. — ISBN 978-5-
9275-2397-9. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: https://www.iprbookshop.ru/87427.html (дата обращения: 15.04.2021). — Режим доступа: для авторизир. пользователей - https://www.iprbookshop.ru/87427.html

3. Пентус, А. Е. Математическая теория формальных языков : учебное пособие / А. Е. Пентус, М. Р. Пентус. — 3-е изд. — Москва : ИнтернетУниверситет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020.
— 218 с. — ISBN 978-5-4497-0662-1. — Текст : электронный // Электроннобиблиотечная система IPR BOOKS : [сайт]. — URL: https://www.iprbookshop.ru/97548.html (дата обращения: 15.04.2021). — Режим доступа: для авторизир. пользователей - https://www.iprbookshop.ru/97548.html

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. —ISBN 978-5-292-04613-4.

— Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт].
— URL: https://www.iprbookshop.ru/99047.html (дата обращения: 15.04.2021). — Режим доступа: для авторизир. пользователей - https://www.iprbookshop.ru/99047.html

5. Малявко, А. А. Формальные языки и компиляторы : учебник / А. А. Малявко. — Новосибирск : Новосибирский государственный технический университет, 2014. — 431 с. — ISBN 978-5-7782-2318-9. — Текст : электронный //
Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: https://www.iprbookshop.ru/47725.html (дата обращения: 15.04.2021). — Режим доступа: для авторизированных пользователей - https://www.iprbookshop.ru/47725.html.

Приложения

Исходный код проекта представлен в репозитории https://github.com/B4N4N41C/Coursework1.git