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»
(МИВлГУ)

Факультет ИТР

Кафедра ПИн

КУРСОВАЯ РАБОТА

По Теория автоматов и формальных языков

Тема Транслятор подмножества языка VB

Руководитель

Кульков Я.Ю.
(фамилия, инициалы)

(оценка)

(подпись) (дата)

Члены комиссии

Студент ПИн-121
(группа)

Ермилов М.В.
(фамилия, инициалы)

(подпись) (Ф.И.О.)

(подпись) (Ф.И.О.)

(подпись) (дата)

Муром 2023

В данной курсовой работе описан процесс создания транслятора с подмножества языка Visual Basic. Для создания приложения использовался язык программирования C#, а также среда разработки Visual Studio 2022. Данный транслятор создан под ОС Windows.

This course work describes the process of creating a translator from a subset of the Visual Basic language. To create the application, the C# programming language was used, as well as the Visual Studio 2022 development environment. This translator was created for the Windows operating system.

Содержание

Введение

С появлением первых компьютеров программисты серьезно задумались над проблемой кодирования компьютерных программ. Уже с конца 40-х годов стали появляться первые примитивные языки программирования высокого уровня. В них решаемая задача записывалась в виде математических формул, а затем переводилась символ за символом в коды. В дальнейшем специальные программы превращали эти коды в двоичный машинный код. Такой программой является транслятор.

Транслятор – это программа, которая переводит входную программу на исходном (входном) языке в эквивалентную ей выходную программу на результирующем (выходном) языке. То есть транслятор – это часть программного обеспечения, который представляет собой набор машинных команд и данных и выполняется компьютером. Все составные части транслятора представляют собой фрагменты и модули программы со своими входными и выходными данными.

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

Целью данной курсовой работы является создание такого транслятора с подмножества языка Visual Basic.Чтобы создать транслятор, необходимо прежде всего построить грамматику входного языка, произвести лексический, синтаксический анализ, а также совершить разбор арифметических и логических выражений.

Актуальность данной темы заключается в учебных целях. Понять, как происходит процесс преобразования программы в программу на другом языке, как выполняется лексический анализ, как формируется таблица лексем, как осуществляется синтаксический разбор, узнать и реализовать метод разбора сложных логических выражений.

1 Анализ технического задания

В данной курсовой работе необходимо разработать транслятор с подмножества языка Vusual Basic. Программа будет разрабатываться в среде разработки Visual Studio 2022 с использованием высокоуровневого объектно-ориентированного языка программирования C#. Выбор именно этих инструментов для разработки указанного приложения основывается на том, что Visual Studio 2022 обладает удобным дизайнером форм, который позволяет располагать на форме различные компоненты, начиная с кнопок и текстовых полей, заканчивая таблицами и изображениями. Язык программирования C# выбран по следующим причинам: он бесплатный, а также ориентирован на платформу Windows.

Для создания графической интерфейса приложения будет использоваться технология Windows Forms. Эта технология имеет следующие особенности:

– наличие удобного дизайнера форм в среде разработки Microsoft Visual Studio 2022;

– наличие в .NET Framework достаточного набора графических компонентов для реализации пользовательского интерфейса, обеспечивающего требуемый функционал программы.

К разрабатываемому приложению были выдвинуты следующие требования:

– обеспечить развернутую диагностику ошибок;

– реализовать класс транслятора;

– реализовать синтаксический разбор - на основе LR(k)-грамматик;

– выполнить разбор логических выражений методом Дейкстры;

– в языке поддерживаются: у идентификаторов 8 символов значащие; не менее 3-х директив описания переменных; простой арифметический оператор;

сложное логическое выражение; оператор цикла DO WHILE … LOOP

Все необходимые разборы будут выполняться на заготовленном заранее фрагменте кода. Ниже представлен пример фрагмента кода:

```
Dim a as integer
b=1
do while (a < 10 or b > c)
 b=b+a
loop
```

В начале работы будет написана часть программы, которая будет производить лексический анализ и классификацию лексем. Это поможет разобрать на лексические единицы анализируемый код, что в дальнейшем позволит проще производить синтаксический анализ.

Далее необходимо будет построить грамматику языка, для основы которой будет браться вышеуказанный фрагмент кода, но также грамматика должна учитывать возможность вложенности условных операторов, операторов объявления переменных и операторов присваивания в условном операторе. На основе построенной грамматики будет построена новая грамматика, которая должна будет принадлежать к классу LR(k). Эта грамматика требуется для реализации метода рекурсивного спуска.

На основе LR(k)-грамматики будет реализован синтаксический анализ. Данный анализ будет проводиться методом рекурсивного спуска. Каждому правилу грамматики будет соответствовать своя подпрограмма (метод), которая будет последовательно проверять текущие лексемы с ожидаемыми. Если результат будет отличаться от ожидаемого, то анализ прекратиться, и будет выведена соответствующая ошибка. Метод рекурсивного спуска будет также проверять правильность простого арифметического оператора.

Заключительным этапом работы будет являться разбор сложных логических выражений методом Бауэра-Замельзона. Результатом этого разбора будет являться матрица логического оператора.

2 Описание грамматики языка

2.1 Описание языка Visual Basic

Visual Basic - язык программирования, разработанный компанией Microsoft. Язык Visual Basic унаследовал дух, стиль и отчасти синтаксис своего предка — языка BASIC, у которого есть немало диалектов. В то же время Visual Basic сочетает в себе процедуры и элементы объектно-ориентированных и компонентно-ориентированных языков программирования. Интегрированная среда разработки VB включает инструменты для визуального проектирования пользовательского интерфейса, редактор кода с возможностью IntelliSense и подсветкой синтаксиса, а также инструменты для отладки приложений.

Visual Basic также является хорошим средством быстрой разработки (RAD) приложений баз данных для операционных систем семейства Microsoft Windows. Множество готовых компонентов, поставляемых вместе со средой, призваны помочь программисту сразу же начать разрабатывать бизнес-логику бизнес-приложения, не отвлекая его внимание на написание кода запуска программы, подписки на события и других механизмов, которые VB реализует автоматически.

2.2 Оператор объявления переменных

Объявление переменных – процесс введения новых переменных в программу. Объявление включает в себя указание идентификатора, типа и, при необходимости, указание начального значения.

Структура оператора в языке VB:

Dim <Названия переменных> As <Тип> [= Значение]

В блоке <Названия переменных> могут писаться одна переменная или список переменных, разделённый между собой запятой. Блок [= Значение] необязателен и может быть представлен переменной, значением, согласно типу,

или выражением.

2.3 Оператор присваивания

Присваивание - это запись данных в участок памяти компьютера, отведенной для значения величины, тех данных, которые хранятся в другом участке памяти компьютера, где записано значение величины, то есть оператор присваивания служит для изменения значения переменной.

Структура оператора: <переменная>=<выражение>

Слева от знака «=» в операторе присваивания записывается имя той переменной, которой нужно присвоить новое значение, а справа - выражение, которое может быть представлено как переменная, определенное значение или арифметическое выражение, результат которого необходимо вычислить. Значение, хранящееся в блоке <выражение>, присваивается переменной.

2.4 Условный оператор

Оператор цикла DO WHILE … LOOP в языке Visual Basic позволяет выполнять определенные команды в цикле до тех пор, пока некоторое логическое выражение (условие) принимает значение «истина».

Структура оператора цикла DO WHILE … LOOP следующая:

Do While <условие><команды> Loop

При выполнении оператора цикла DO WHILE ... LOOP сначала проверяется логическое выражение <условие>. Если оно истинно, то выполняются команды, находящиеся между операторами DO WHILE и LOOP, а затем проверяется условие снова. Цикл продолжается до тех пор, пока условие остается истинным. Если условие становится ложным, то выполнение цикла прекращается и управление передается следующей инструкции после оператора LOOP.

Условие может быть представлено простым или сложным логическим

выражением.

2.5 Грамматика языка

Для разработки решающего автомата необходимо продумать грамматику подмножества языка Visual Basic, включающую:

- Терминалы;
- Нетерминалы;
- Правила языка;
- Начальное правило языка.

В результате анализа вышеописанных конструкций языка Visual basic была создана следующая грамматика:

G = {T, N, P, <программа>}

T = {Dim, as, integer, double, string, do, while, loop, =, >, *, +, /, -, <, id, lit, expr, \n}

N = {<программа>, <спис_опер>, <опер>, <тип>, <знак>, <операнд>, <цикл>, <матем>, <присв>, <спис_перем>, <перем>}

P = {

<программа>::= <спис_опер>

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

<опер>::=<присв> | <перем> | <цикл>

<перем>::=Dim <спис_перем> as <тип>

<спис_перем>::=id, <спис_перем> | id

<тип>::=integer | double | string

<присв>::=id=<матем>

<матем>::=<операнд><знак><матем> | <операнд>

<операнд>::=id | lit

<знак>::=+| -| * | /

<цикл>::=do while exp\n<спис_опер>\nloop

}

Здесь знак /- отвечает за конец строки, lit – литерал, id – идентификатор, expr-это функция для анализа сложного логического выражения.

2.6 Грамматика языка для синтаксического анализа

В соответствии с техническим заданием синтаксическим разбор лексем следует выполнять нисходящим анализатором на основе LR(k)-грамматики.

Таблица 1 – Решающая таблица восходящего анализатора

состояние	Предыдущее состояние	Правила граматики	переход
0		<программа>::=*<спис_опер>	1
1	0,3,28	<спис_опер>::=*<опер>\n<спис_опер> <спис_опер>::=*<опер> <опер>::=*<присв> <опер>::=*<перем> <опер>::=*<цикл> <перем>::=*Dim <спис_перем> as <тип> <присв>::=*id=<матем> <цикл>::=*do while exp\n<спис_опер>\nloop	2 2 - - - 16 4 25
2	1	<спис_опер>::=<опер>*\n<спис_опер> <спис_опер>::=<опер>*	3 -
3	2	<спис_опер>::=<опер>\n*<спис_опер>	1
4	1	<присв>::=id*=<матем>	5
5	4,11	<присв>::=id=*<матем> <матем>::=*<операнд><знак><матем> <матем>::=*<операнд>	6 7 -

		<операнд>::=*id <операнд>::=*lit	8 9
6	5	<присв>::=id=<матем>*	-
7	5	<матем>::=<операнд>*<знак><матем> <знак>::=*+ <знак>::=*- <знак>::=** <знак>::=*/	11 12 13 14 15
9	5	<операнд>::=id*	-
10	5	<операнд>::=lit*	-
11	7	<матем>::=<операнд><знак>*<матем>	5
12	7	<знак>::=+*	-
13	7	<знак>::=-*	-
14	7	<знак>::=**	-
15	7	<знак>::=/*	-
16	1	<перем>::=Dim *<спис_перем> as <тип> <спис_перем>::= *id, <спис_перем> <спис_перем>::= *id	17 23
17	16	<перем>::=Dim <спис_перем>*as <тип>	18
18	17	<перем>::=Dim <спис_перем> as *<тип> <тип>::=*integer <тип>::=*double <тип>::=*string	19 20 21 22
19	18	<перем>::=Dim <спис_перем> as <тип>*	-
20	18	<тип>::=integer*	-
21	18	<тип>::=double*	-
22	18	<тип>::=string*	-
23	16,24	<спис_перем>::= id*, <спис_перем>	24

		<спис_перем>::= id*	-
24	23	<спис_перем>::= id, *<спис_перем>	16
25	1	<цикл>::=do *while exp\n<спис_опер>\nloop	26
26	25	<цикл>::=do while *exp\n<спис_опер>\nloop	27
27	26	<цикл>::=do while exp*\n<спис_опер>\nloop	28
28	27	<цикл>::=do while exp\n*<спис_опер>\nloop <спис_опер>::=*<опер>\n<спис_опер> <спис_опер>::=*<опер>	29 1 1
29	28	<цикл>::=do while exp\n<спис_опер>*\nloop	30
30	29	<цикл>::=do while exp\n<спис_опер>\n*loop	31
31	30	<цикл>::=do while exp\n<спис_опер>\nloop*	-

В результате построения решающей таблицы было выявлено, что вышеописанная грамматика относится к классу LR(2), так как в одном правиле необходимо знать две лексемы, чтобы анализатор понял, что ему делать дальше.

3 Разработка архитектуры системы и алгоритмов

Для реализации транслятора с подмножества языка Visual Basic необходимо реализовать лексический анализатор, синтаксический анализатор, а после этого анализатор сложных логических выражений.

3.1 Работа лексического анализатора

Алгоритм работы лексического анализатора можно представить в виде диаграммы действий (Рисунок 1).

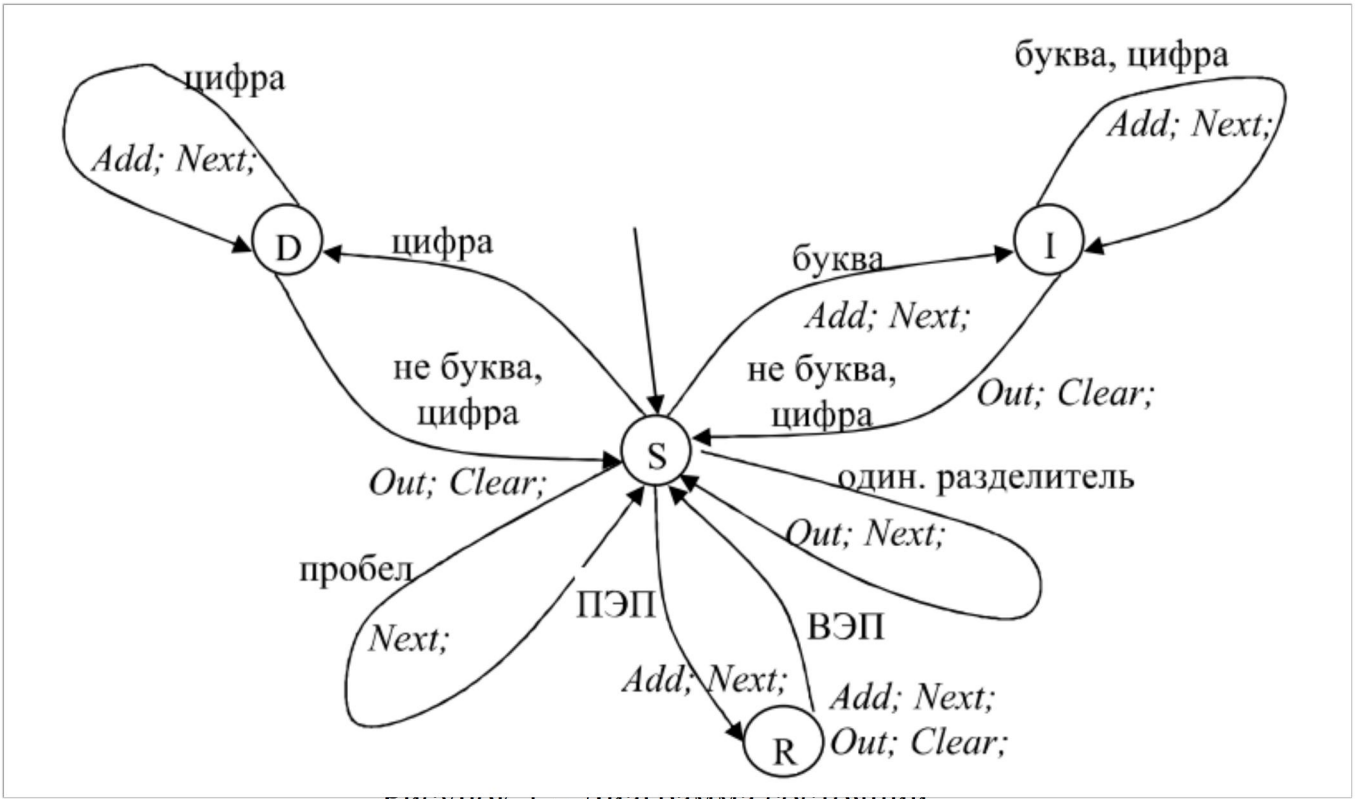

Рисунок 1 – Диаграмма состояний

Алгоритм лексического анализатора:

1. Объявляем текущим начальное состояние S диаграммы.

2. До тех пор, пока не будет достигнуто конечное состояние диаграммы на последнем элементе входного потока или получена ошибка лексического анализа, считываем очередной символ анализируемой строки и переходим из текущего

состояния диаграммы в другое по дуге, помеченной этим символом, выполняя при этом соответствующие действия. Состояние, в которое попадаем, становится текущим.

3. Выходной поток формируется вызовом подпрограммы out. Out (лексема, тип) – подпрограмма (метод), которая выдаёт информацию о накопленной лексеме в выходной поток. Тип задаётся веткой диаграммы состояний по которой была собрана лексема. Данная подпрограмма также проверяет длину идентификатора и значение литерала. Если порог максимальной длины/значения превышен, то выводится сообщение с ошибкой, работа лексического анализатора прекращается.

Результатом работы лексического анализатора является перечень всех найденных в анализируемом коде лексем. Его можно представить в виде таблицы пар (лексема, её предварительный тип) или списка. Предварительными типами являются идентификатор (I), разделитель (R), литерал (D). После происходит разделение идентификаторов на ключевые слова и переменные и формирование следующих таблиц: таблица ключевых слов, таблица переменных, таблица разделителей, таблица литералов и таблица, содержащая в себе строки вида (номер таблицы определенного типа лексемы; номер лексемы в таблице, соответствующей её типу).

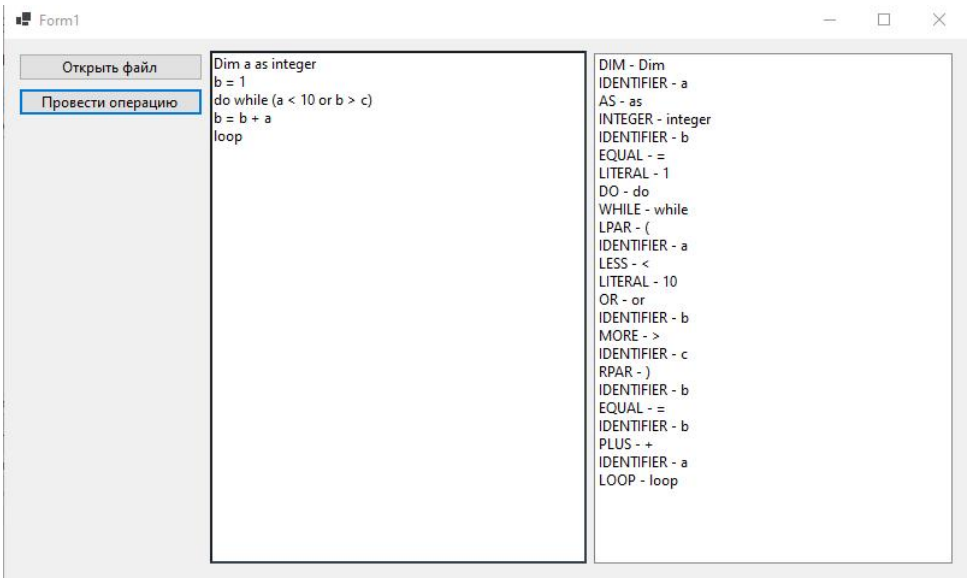

Рисунок 2 – Результат лексического анализа

3.2 Синтаксический анализатор

Для реализации синтаксического анализатора был реализован класс SyntacticAnalysis, который содержит поля и методы для реализации метода рекурсивного спуска. Каждый метод в классе реализует одно единственное правило, соответствующее решающей таблице.

Метод рекурсивного спуска – это нисходящиий метод синтаксического анализа, в котором входной поток лексем обрабатывается выполнением ряда рекурсивных процедур. Каждому нетерминалу грамматики соответствует одна подпрограмма, которая распознаёт цепочку, порождаемую этим нетерминалом. Последовательность вызовов процедур неявно определяет дерево разбора. Метод рекурсивного спуска применим в классе LR(k)-грамматик.

При разработке подпрограмм, необходимо придерживаться следующих правил:

1. Каждому нетерминалу соответствует подпрограмма. Если одному нетерминалу в левой части правила соответствует более одной правой части, то первый шаг – выбор нужного правила из списка альтернативных, для чего используется информация из второй колонки решающей таблицы, иначе первый шаг – пропустить. На этом этапе программируется анализ текущего элемента входного потока и следующих за ним (но не более k) и в случае совпадения элементов с эталонным набором из таблицы, программируется ветка, соответствующая этому эталону (см. правила 2 и 3). Смены текущего элемента при этом не происходит. Если нет совпадения ни с одним из возможных эталонных вариантов, то фиксируем ошибку и выходим из подпрограммы.

2. Если текущим символом правой части правила грамматики является нетерминал, ему соответствует вызов подпрограммы, имя которой совпадает с этим нетерминалом и которая распознаёт цепочку, порождаемую им.

3. Если текущим символом правой части правила грамматики является терминал, то проверяем текущий элемент цепочки на совпадение с этим

терминалом. Если совпадения нет – фиксируем ошибку и выходим из подпрограммы, иначе перемещаемся к следующему символу входного потока (текущим становится следующий).

3.3 Транслятор логических выражений

Транслятор логических выражений (правило expr) в соответствии с техническим заданием следует реализовывать методом Бауэра-Замельзона.

Алгоритм работы транслятора следующий:

В методе используются два стека и таблица переходов. Один стек, обозначим его E, используется для хранения операндов, другой стек – T, для хранения знаков операций. Над стеком E выполняются две операции: K(id) – выбрать элемент с именем id из входного потока, положить на вершину стека E, перейти к следующему элементу входного потока; K(OP) – извлечь два верхних операнда из стека E, записать тройку: (OP, операнд, операнд) в матрицу арифметического оператора; записать результат на вершину стека E. Над стеком T выполняются операции согласно таблице переходов (Рисунок 3).

	$	(	> < = =	+ - OR	* / AND	)
ε	D6	D1	D1	D1	D1	D5
(	D5	D1	D1	D1	D1	D3
< > = =	D4	D1	D2	D1	D1	D4
+ - OR	D4	D1	D4	D2	D1	D4
* / AND	D4	D1	D4	D4	D2	D4

Рисунок 3 – таблица переходов

Описание правил в таблице переходов:

D1. Записать OP в стек T и читать следующий символ строки.

D2. Удалить OP1 из стека T и генерировать команду K(OP1); Записать OP в стек T и читать следующий символ строки.

D3. Удалить OP1 из стека T и читать следующий символ строки.

D4. Удалить OP1 из стека T и генерировать команду K(OP1);

D5. Ошибка в выражении. Конец разбора.

D6. Успешное завершение разбора.

В соответствии с описанным алгоритмом был реализован транслятор выражений, корректно разбирающий простые и сложные логические выражения. Матрица логического оператора в результате разбора выражения (a < 3) And (b <> 5) выглядит следующим образом (Рисунок 4):

	Операн 1	Действ	Операнд 2	Результат
▶	a	MORE	12	m0
	b	MORE	12	m1

Рисунок 4 – Разбор сложного логического выражения

4 Методика испытаний

Для проверки работы анализатора подмножества языка следует проверить работу лексического, синтаксического анализаторов и транслятора арифметических выражений.

4.1 Проверка лексического анализатора

Для проверки лексического анализатора необходимо ввести код на языке Visual Basic и начать анализ кода. Если были введены корректные лексемы, то высветиться окно с сообщением об успешном выполнении лексического анализа, а также станет доступна форма с данными о лексемах и её типах (Рисунок 5-6).

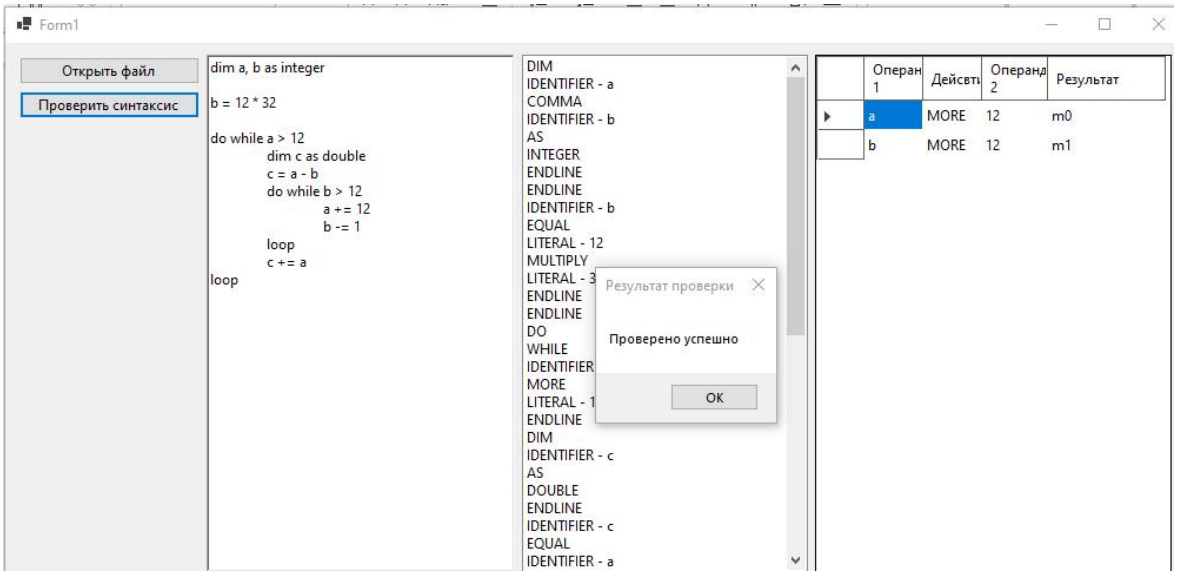

Рисунок 5 – Сообщение об успешном выполнении лексического анализа

Если же будет введён символ, который отсутствует в грамматике языка, то выведется сообщение с ошибкой, форма с данными о лексемах и её типах будет недоступна (Рисунок 6).

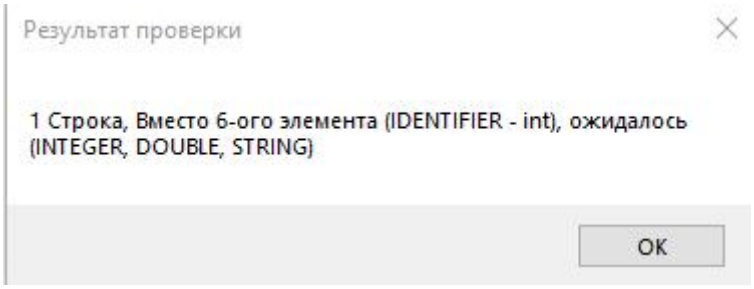

Рисунок 6 – Ошибка в лексическом анализе

4.2 Проверка синтаксического анализатора

Если лексический анализ успешно произведён, то после него начнётся синтаксический анализ. Если данный анализ не выполниться, то выведется соответствующее сообщение с описанием проблемы (Рисунок 7).

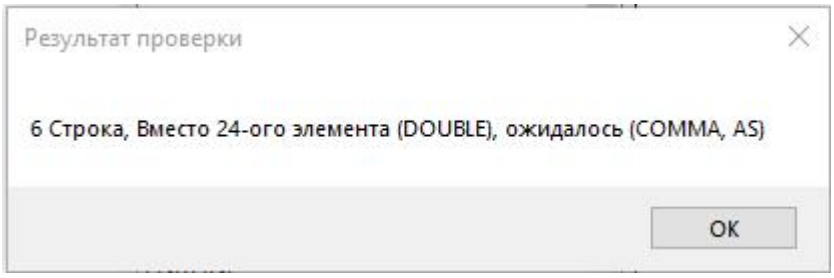

Рисунок 7 - сообщение об ошибке

5 Руководство пользователя

При запуске exe-файла открывается главное меню программы. Пользователь, может открыть уже существующий файл, и отредактировать его, или написать свой собственный код (Рисунок 8)

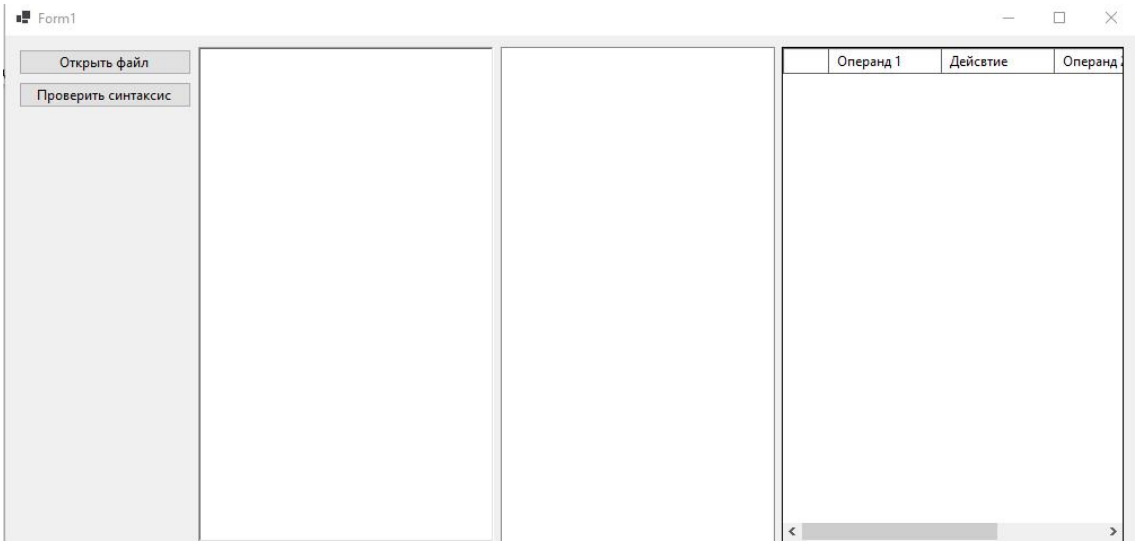

Рисунок 8 - Начальное окно

После добавления кода, пользователь может нажать кнопку, «проверить синтаксис», после чего проведется разбиение кода на ликсемы, анализ кода, и анализ логических выражений, с выводом результата анализа в крайнюю панель (Рисунок 9).

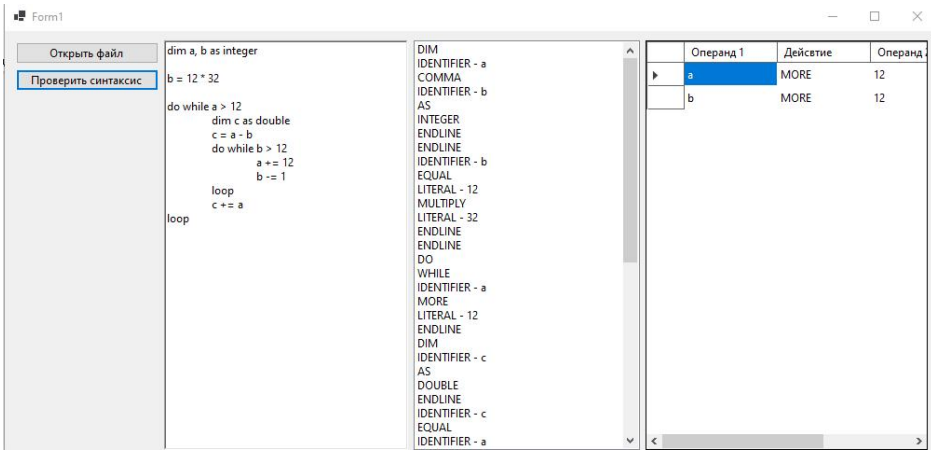

Рисунок 9 - Интерфейс программы, после проверки кода

6 Руководство программиста

Исходный код разработанного приложения содержит в себе множество классов, каждый из которых хранит в себе множество методов. Руководство включает в себя описание классов, методов и полей программы.

6.1 enum TokenType

Это набор типов, который соответсвует лексемам языка Visual Basic, и в котором храняться все необходимые данные, для описания оператора do while.

6.2 class Token

Класс, которые хранит в себе статичные поля, как:

Dictionary<string, TokenType> SpecialWords - Хранит все зарезервированные слова, в данном языке

Dictionary<string, TokenType> SpecialSymbols - Хранит в себе, все специальные символы языка

Так-же, данный класс хранит в себе статичные методы, такие как:

bool isIdentifier(string word) - Проверяет, может ли быть, поступаемая строка, названием переменной

bool IsSpecialWord(string word) - Проверяет, является ли поступаемая строка, зарезервированным словом

bool IsSpecialSymbol(string ch) - Проверяет, является ли поступаемая строка, специальным символом

Так-же, данный класс хранит в себе поля, для создания объекта, такие как:

TokenType Type

string Value - Хранит в себе, либо значение литерала, либо название индетификатора

Данный класс, так-же имеет конструкт, на вход которого поступает строка, по которой определяется тип данной лексемы

6.3 class generation

Данный класс, хранит в себе 2 статичных метода, для преобразования поступаемой строки в коллекцию объектов с типом Token

string[] Split(string code) - Разбивает код, на пробелы, табуляции и специальные символы описанные в Token.SpecialSymbols

Token[] Tokens(string[] keys) - Метод, который, по полученному массиву, из раннее описанного метода, преобразует в массив объектов с типом Token

6.4 class FormMain

Данный класс, содержит в себе, описание работы с визуальной частью, и имеет следующие методы:

OpenFile(object sender, EventArgs e) - Метод, который открывает выбранный файл, и записывает его в программу

button3_Click(object sender, EventArgs e) - Метод, который запускает разбиение текста на массивы объектов Token, и запускает все соопутсвующие проверки

6.5 class LR

Данный класс, реализует в себе разбор полученного массива, и проверку его на соответсвие выше приложенным таблицам. Так-же имеет в себе, такой метод, как:

Check() = Метод, который вызывает проверку, в случае ошибки, вылезает исключение, с описением проблемы

Так-же, данный класс, имеет свойство Troyka, которая хранит матрицу действий, разложенныю по методу Бауэра-Замельзона

6.6 class Bower

Имеет в себе функционал, по разбору сложных логических выражений. Имеет следующий метод: Start() - который запускает проверку. Так-же, имеет свойство, как troyka, которое хранит матрицу, как в LR.Troyka, и имеет свойство Lastindex.

6.7 struct Troyka

Структура, которая хранит в себе два операнда и оператор действия

Заключение

В процессе выполнения курсового проекта был создан транслятор, способный переводить подмножество языка Visual Basic согласно требованиям, изложенным в ТЗ. Для создания транслятора использовались среда разработки Visual Studio и объектно-ориентированный язык программирования C#.

Для реализации транслятора был произведен лексический анализ, в результате которого были получены списки лексем и их типов. Затем была построена грамматика языка, приведенная к классу LR(k), и реализован синтаксический анализ. Для разбора сложных логических выражений был использован метод Бауэра-Замельзона.

Если потребуется расширить функционал программы, можно добавить проверку на соблюдение семантических соглашений языка, что позволит оценить состояния, которые не могут быть проверены на этапе синтаксического анализа. Программа занимает небольшой объем памяти и не требует высоких системных требований для работы.

Список литературы

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

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

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

Приложение А. Ссылка на репозиторий

Программный код проекта расположен в следующем репозитории:
https://github.com/m9agrest/TranslatorVB_While