МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий
Кафедра Программной инженерии
Специальность 1-40 01 01 Программное обеспечение информационных технологий
Специализация Программирование интернет-приложений

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:

«Разработка компилятора MMV-2020»

Выполнил студент Малиновский Максим Витальевич
(Ф.И.О.)

Руководитель проекта преп.-стаж. Котович Дмитрий Витальевич
(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Пацей Н.В.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты преп.-стаж. Котович Дмитрий Витальевич
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой _____

Содержание

Введение

В данном курсовом проекте поставлена задача разработки собственного языка программирования и транслятора для него. Название языка – MMV-2020. Написание транслятора будет осуществляться на языке C++, при этом код на языке MMV-2020 будет транслироваться в язык C#.

Задание на курсовой проект можно разделить на следующие задачи:

1. Разработка спецификации языка MMV-2020;
2. Разработка лексического анализатора;
3. Разработка синтаксического анализатора;
4. Разработка семантического анализатора;
5. Разбор арифметических выражений;
6. Разработка генератора кода;
7. Тестирование транслятора.

1 Спецификация языка программирования

1.1 Характеристика языка программирования

Язык MMV-2020 – это универсальный, строго типизированный, интерпретируемый язык, не имеющий элементов ООП.

1.2 Определение алфавита языка программирования

В алфавите языка MMV-2020 используется кодировка ASCII, таблица которой представлена на рисунке 1.1.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00 □	01 □	02 □	03 □	04 □	05 □	06 □	07 □	08 □	09 □	0A □	0B □	0C □	0D □	0E □	0F □
10	10 □	11 □	12 □	13 □	14 □	15 □	16 □	17 □	18 □	19 □	1A □	1B □	1C □	1D □	1E □	1F □
20	20	21 !	22 “	23 #	24 $	25 %	26 &	27 (	28)	29 '	2A *	2B +	2C ,	2D -	2E .	2F /
30	30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7	38 8	39 9	3A :	3B ;	3C <	3D =	3E >	3F ?
40	40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G	48 H	49 I	4A J	4B K	4C L	4D M	4E N	4F O
50	50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W	58 X	59 Y	5A Z	5B [	5C \	5D]	5E ^	5F _
60	60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g	68 h	69 i	6A j	6B k	6C l	6D m	6E o	6F p
70	70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w	78 x	79 y	7A z	7B {	7C |	7D }	7E ~	7F □

Рисунок 1.1 Алфавит входных символов

Символы, используемые на этапе выполнения: [a…z], [A…Z], [0…9], символы пробела, табуляции и перевода строки, спецсимволы: [ ] ( ) , ; {} + - / * > <, &, ., `,~, %, ?, #.

1.3 Применяемые сепараторы

Символы-сепараторы служат в качестве разделителей цепочек языка во время обработки исходного текста программы с целью разделения на токены. Они представлены в таблице 1.1.

Таблица 1.1 Символы-сепараторы

Символ(ы)	Назначение
‘пробел’	Разделитель цепочек. Допускается везде кроме названий идентификаторов и ключевых слов
{...}	Блок функции
(...)	Приоритет арифметических операций
&	Разделитель параметров функций
+ - */	Арифметические операции
> < ` ~ # ?	Логические операции (операции сравнения: больше, меньше, меньше либо равно, больше либо равно, проверка на равенство, на неравенство), используемые в условной конструкции.
.	Разделитель программных конструкций
=	Оператор присваивания
[...]	Блок фактических или формальных параметров функции

1.4 Применяемые кодировки

Для написания программ язык MMV-2020 использует кодировку ASCII, содержащую английский алфавит, а также некоторые специальные символы, такие как [] () {} + - / * %> <, &, ., `,~, %., ?, #.

1.5 Типы данных

В языке MMV-2020 реализованы два фундаментальных типа данных: целочисленный и строковый. Описание типов приведено в таблице 1.2.

Таблица 1.2 Типы данных языка MMV-2020

Тип данных	Характеристика
1	2
Целочисленный тип данных **int**	Фундаментальный тип данных. Используется для работы с числовыми значениями. В памяти занимает 1 байта. Максимальное значение: 127. Минимальное значение: -128. Инициализация по умолчанию: значение 0. Поддерживаемые операции: + бинарный) – оператор сложения; - (бинарный) – оператор вычитания; * (бинарный) – оператор умножения; / (бинарный) – оператор деления; = (бинарный) – оператор присваивания; Библиотечные функции. В качестве условия условного оператора поддерживаются следующие логические операции: > (бинарный) – оператор «больше»; < (бинарный) – оператор «меньше»; ? (бинарный) – оператор проверки на равенство; ` (бинарный) – оператор «меньше либо равно»; ~ (бинарный) – оператор «больше либо равно»; # (бинарный) – оператор проверки на неравенство.

1	2
Строковый тип данных **str**	Фундаментальный тип данных. Используется для работы с символами, каждый из которых занимает 1 байт. Максимальное количество символов – 255. Инициализация по умолчанию: строка нулевой длины “”.
Логический тип данных bool	Фундаментальный тип данных. В памяти занимает 1 байт. Автоматическая инициализация false. Максимальное значение true

1.6 Преобразование типов данных

Преобразование типов данных в языке MMV-2020 не предусмотрено. Попытка преобразования типов данных приведет к семантической ошибке.

1.7 Идентификаторы

Общее количество идентификаторов ограничено максимальным размером таблицы идентификаторов. Идентификаторы должны содержать только символы нижнего регистра латинского алфавита. Максимальная длина идентификатора равна пяти символам. Идентификаторы не должны совпадать с ключевыми словами. Типы идентификаторов: имя переменной, имя функции, параметр функции.

1.8 Литералы

С помощью литералов осуществляется инициализация переменных. Типы литералов языка MMV-2020 представлены в таблице 1.3.

Таблица 1.3 Литералы

Литералы	Пояснение
1	2

1	2
Целочисленные литералы	Последовательность цифр 0…9
Строковые литералы	Набор символов (от 1 до 255), заключённых в двойные кавычки
Логические литералы	2 логических константы - true(истина) и false(ложь)

Ограничения на строковые литералы языка MMV-2020: внутри литерала не допускается использование одинарных и двойных кавычек. Ограничения на целочисленные литералы: не могут быть отрицательными.

1.9 Объявление данных

Для объявления переменной используется ключевое слово **var**, после которого указывается тип данных и имя идентификатора. Допускается инициализация при объявлении.

Пример объявления числового типа с инициализацией:

var int num = 1.

Пример объявления переменной символьного типа с инициализацией:

var str str1= “hello world”.

Для объявления функций используется ключевое слово **function**, после которого указывается тип функции, а после – имя функции. Далее список параметров и тело функции.

1.10 Инициализация данных

При объявлении переменной допускается инициализация данных. При этом переменной будет присвоено значение литерала или идентификатора, стоящего справа от знака равенства. Объектами-инициализаторами могут быть только идентификаторы или литералы. При объявлении без инициализации предусмотрены значения по умолчанию: значение 0 для типа **int,** строка длины 0 (“”) для типа **str,** false для типа **bool**.

1.11 Инструкции языка

Инструкции языка MMV-2020 представлены в таблице 1.4.

Таблица 1.4 Инструкции языка MMV-2020

Инструкция	Реализация
1	2
Объявление переменной	**var** <тип данных> <идентификатор>.
Объявление переменной с явной инициализацией	**var** <тип данных> <идентификатор> = <значение>. Значение – инициализатор конкретного типа. Может быть только литералом или идентификатором
Возврат из функции или процедуры	Для функций, возвращающих значение: **return** [идентификатор/литерал].
Вывод данных	**output [**идентификатор/литерал].
Вызов функции	<идентификатор функции> [(<список параметров>].
Присваивание	<идентификатор> = <выражение>. Выражением может быть идентификатор, литерал, или вызов функции соответствующего типа. Для целочисленного типа выражение может быть дополнено арифметическими операциями с любым количеством операндов с использованием скобок. Для строкового типа выражение может быть только идентификатором, литералом или вызовом функции, возвращающей значение строкового типа.

1.12 Операции языка

В языке MMV-2020 предусмотрены следующие операции с данными. Приоритетность операции умножения и деления выше приоритета операций сложения и вычитания. Для установки наивысшего приоритета используются круглые скобки. Операции языка представлены в таблице 1.5.

Таблица 1.5 Операции языка MMV-2020

Тип оператора	Оператор
Арифметические	1. + – сложение 2. - – вычитание 3. * – умножение 4. / – деление 5. % – остаток от деления 6. = – присваивание
Строковые	1. = – присваивание 2. + - конкатенация
Логические	1. > – больше 2. < – меньше 3. ? – проверка на равенство 4. # – проверка на неравенство 5. ` меньше либо равно 6. ~ больше либо равно

1.13 Выражения и их вычисление

Вычисление выражений – одна из важнейших задач языков программирования. Всякое выражение составляется согласно следующим правилам:

1. Допускается использовать скобки для смены приоритета операций;
2. Выражение записывается в строку без переносов;
3. Использование двух подряд идущих операторов не допускается;
4. Допускается использовать в выражении вызов функции, вычисляющей и возвращающей целочисленное значение.

1.14 Конструкции языка

Программа на языке MMV-2020 оформляется в виде функций пользователя и главной функции.

Программные конструкции языка представлены в таблице 1.6.

Таблица 1.6 Программные конструкции языка MMV-2020

Конструкция	Реализация
1	2

Главная функция	**head**
	{
	…
	}

Внешняя функция	**function**<тип данных> <идентификатор>
	[<тип> <идентификатор> &…]
	{
	...
	return[идентификатор/литерал].
	}

1	2
Условная конструкция	**check [** <идентификатор1> <оператор> <идентификатор2>] { … } **otherwise** { … } <идентификатор1>, <идентификатор2> - идентификаторы, литералы целочисленного типа или логического (одинакового типа). <оператор> - один из операторов сравнения, устанавливающий отношение между двумя операндами и организующий условие данной конструкции.

1.15 Области видимости идентификаторов

Область видимости: сверху вниз (как и в C++). Переменные, объявленные в одной функции, недоступны в другой. Все объявления и операции с переменными происходят внутри какого-либо блока. Внутри условной конструкции объявление переменных запрещено.

Все идентификаторы являются локальными и обязаны быть объявленными внутри какой-либо функции. Глобальных переменных нет. Параметры видны только внутри функции, в которой объявлены.

1.16 Семантические проверки

В языке программирования MMV-2020 выполняются следующие семантические проверки:

– Закрытие строкового литерал;
– Отсутствует точка входа head;
– Несколько точек входа head;
– Превышение размера строкового литерала;
– Объявление переменной без ключевого слова var;
– Использование необъявленного идентификатора;
– Объявление переменной без указания типа;

– Попытка реализовать уже существующую функцию;
– Попытка переопределить параметр или ошибка области видимости;
– Попытка переопределить переменную или ошибка области видимости;
– Не указан тип функции;
– Использование пустого строкового литерала;
– Тип функции и тип возвращаемого значения должны совпадать;
– Превышение максимально допустимого (2) количество параметров функции;
– Несовпадение типов передаваемых параметров;
– Слишком много аргументов в вызове функции;
– Слишком мало аргументов в вызове функции;
– Несовместимые типы при присваивании;
– Недопустимый целочисленный литерал;
– Типы данных в выражении не совпадают;
– Проверка употребления арифметических операторов;
– Проверка употребления логических операторов.

В языке MMV-2020 присутствует стандартная библиотека, которая подключается автоматически при трансляции исходного кода в язык ассемблера. Содержимое библиотеки и описание функций представлено в таблице 1.8.

Таблица 1.8 Стандартная библиотека языка MMV-2020

Функция	Описание
MaxNum [num1& num2].	Целочисленная функция. Возвращает максимальное из двух значение.
Pow [num1 & num2]	Целочисленная функция. Возводит num1 в степень num2.

Стандартная библиотека написана на языке C#, подключается к транслированному коду на этапе генерации кода. Вызовы стандартных функций доступны там же, где и вызов пользовательских функций.

1.18 Ввод и вывод данных

Вывод данных осуществляется с помощью оператора **output**. Допускается использование оператора **output** с литералами и идентификаторами.

1.19 Точка входа

В языке MMV-2020 каждая программа должна содержать главную функцию (точку входа) **head**, с первой инструкции которой начнётся последовательное выполнение команд программы.

1.20 Препроцессор

Препроцессор, принимающий и выдающий некоторые данные на вход транслятору, в языке MMV-2020 отсутствует.

1.21 Классификация сообщений транслятора

Генерируемые транслятором сообщения определяют степень его информативности, то есть сообщения транслятора должны давать максимально полную информацию о допущенной пользователем ошибке при написании программы. Сообщения транслятора приведены в таблице 1.10.

Таблица 1.10 Классификация ошибок

Номера ошибок	Характеристика
0 – 200	Системные ошибки
200 – 299	Ошибки лексического анализа
300 – 399	Ошибки семантического анализа
600 – 699	Ошибки синтаксического анализа
400-499, 700-999	Зарезервированные коды ошибок

1.22 Контрольный пример

Контрольный пример демонстрирует главные особенности языка MMV-2020: его фундаментальные типы, основные структуры, функции, процедуры, использование функций статической библиотеки. Исходный код контрольного примера представлен в приложении А.

2 Структура транслятора

2.1 Компоненты транслятора, их назначение и принципы взаимодействия

Исходный код, написанный на языке программирования MMV-2020, является для транслятора входными данными.

Как выходные данные используется объектный код и протоколы работы транслятора, описанные в пункте 2.3.

Компоненты транслятора приведены на рисунке 2.1.

Рисунок 2.1 Структура транслятора MMV-2020

Первоначально на вход лексического анализатора передается исходный код. Анализатором проверяется исходный текст на недопустимые символы, выделяет литералы, идентификаторы и ключевые слова, а также формирует таблицы лексем и идентификаторов.

Далее наступает черед синтаксического анализатора, к нему на вход поступает таблица лексем, полученная на этапе лексического анализа. Если программа построена синтаксически правильно, то осуществляется переход к этапу трансляции стоящему далее, при ином раскладе работа транслятора останавливается.

Наборы функций, проверяющие правила на разных этапах работы транслятора представлены в семантическом анализаторе. Продолжение или остановка работы транслятора всецело зависит от критичности возникающих ошибок.

Генерация кода реализуется посредством чистой интерпретации, без создания промежуточного представления кода. В финале происходит генерация кода, во время исполнения которого формируется объектный код.

2.2 Перечень входных параметров транслятора

Для формирования файлов с результатами работы лексического, синтаксического и семантического анализаторов используются входные параметры транслятора, которые приведены в таблице 2.1.

Таблица 2.1 Входные параметры транслятора языка MMV-2020

Входной параметр	Описание параметра	Значение по умолчанию
-in:<путь к in-файлу>	Файл с исходным кодом на языке MMV-2020, имеющий расширение .txt	Не предусмотрено
-log:<путь к log-файлу>	Файл журнала для вывода протоколов работы программы.	Значение по умолчанию: <имя in-файла>.log
-out:<путь к out-файлу>	Выходной файл – результат работы транслятора. Содержит исходный код на языке C#.	Значение по умолчанию: <имя in-файла>.cs

2.3 Перечень протоколов, формируемых транслятором и их содержимое

В ходе работы программы формируются протоколы работы лексического, синтаксического и семантического анализаторов, которые содержат в себе перечень протоколов работы. В таблице 2.2 приведены протоколы, формируемые транслятором и их содержимое.

Таблица 2.2 Протоколы, формируемые транслятором языка MMV-2020

Формируемый протокол	Описание выходного протокола
1	2
Файл журнала, заданный параметром	Файл с протоколом работы транслятора языка программирования MMV-2020. Содержит таблицу

1	2
"-log:"	лексем и таблицу идентификаторов, протокол работы синтаксического анализатора и дерево разбора, полученные на этапе лексического и синтаксического анализа, а также результат работы алгоритма преобразования выражений к польской записи.
Выходной файл, заданный параметром "-out:"	Результат работы программы – файл, содержащий исходный код на языке C#.

3 Разработка лексического анализатора

3.1 Структура лексического анализатора

Исходный код на языке MMV-2020 является входными данными;
Таблицы лексем и идентификаторов являются выходными данными;

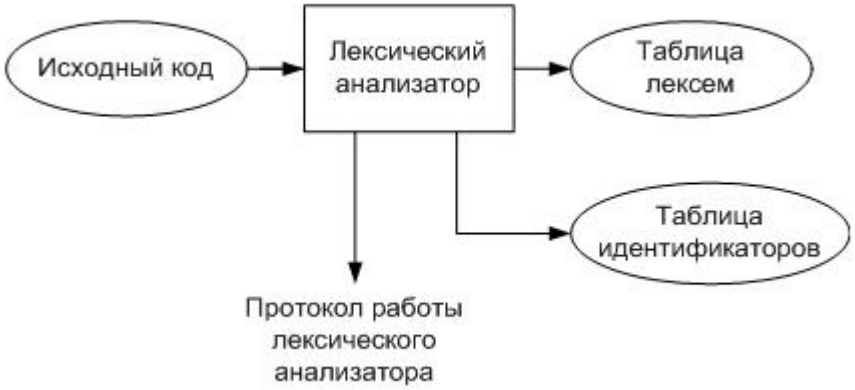

Рисунок 3.1 Структура лексического анализатора

3.2. Контроль входных символов

Для удобной работы с исходным кодом, при передаче его в лексический анализатор, все символы разделяются по категориям. Таблица входных символов представлена на рисунке 3.2, категории входных символов представлены в таблице 3.1.

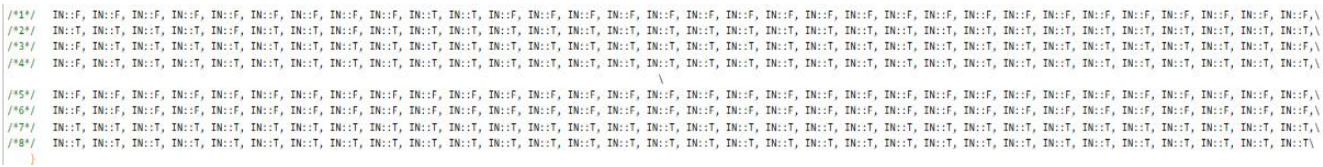

Рисунок 3.2 Таблица контроля входных символов

Таблица 3.1 Соответствие символов и их значений в таблице

Значение в таблице входных символов	Символы
1	2

1	2
Разрешенный	T
Запрещенный	F

3.3 Удаление избыточных символов

Избыточными символами являются символы табуляции и пробелы.

Избыточные символы удаляются на этапе разбиения исходного кода на лексемы перед лексическим анализом.

3.4 Перечень ключевых слов

Перечень цепочек, соответствующих им автоматов и лексем, представлена в приложении Б. Соответствие токенов и лексем приведено в таблице 3.2.

Таблица 3.2 Соответствие токенов и сепараторов с лексемами

Токен	Лексема	Пояснение
1	2	3
var, str, bool	t	Названия типов данных языка.
Идентификатор	i	Длина идентификатора – 8 символов.
Литерал	l	Литерал любого доступного типа.
function	f	Объявление функции.
return	r	Выход из функции.
head	h	Главная функция.
var	v	Объявление переменной.

1	2	3
output	o	Вывод данных.
check	c	Указывает начало условного оператора.
otherwise	u	Ложная ветвь условного оператора
NewLine	n	Оператор вывода символа перевода строки.
.	.	Разделение выражений.
&	&	Разделение параметров функций.
{	{	Начало блока/тела функции.
}	}	Закрытие блока/тела функции.
[	[	Передача параметров в функцию
]	]	Закрытие блока для передачи параметров
=	=	Знак присваивания.
+ - * / %	+ - * / %	Знаки операций.
> < ?	> < &	Знаки логических операторов

1	2	3
` ~ #	! ~ #	

3.5 Основные структуры данных

В приложении В представлены основные структуры данных для хранения таблиц лексем и идентификаторов.

3.6 Принцип обработки ошибок

При возникновении ошибки происходит запись этой ошибки в log-файл и прекращения работы транслятора.

3.7 Структура и перечень сообщений лексического анализатора

Перечень сообщений, формируемых лексическим анализатором в ходе своей работы, представлен в таблице 3.2.

Таблица 3.2 Перечень сообщений лексического анализатора

Код ошибки	Сообщение
1	2
201	Превышен размер таблицы лексем
202	Переполнение таблицы лексем
203	Превышен размер таблицы идентификаторов
204	Переполнение таблицы идентификаторов

1	2
205	Неизвестная последовательность символов

3.8 Параметры лексического анализатора

Текст кода на языке MMV-2020 подается на вход. Параметры не определяют режим работы лексического анализатора.

3.9 Алгоритм лексического анализа

Лексический анализ является первой и наиболее простой фазой трансляции. Алгоритм лексического анализатора заключается в распознавании и разборе цепочек исходного кода. Распознавание цепочек основывается на работе конечных автоматов.

Работу конечного автомата можно проиллюстрировать с помощью графа переходов. Пример графа для цепочки «output» представлен на рисунке 3.2, где S0 – начальное, а S6 – конечное состояние автомата.

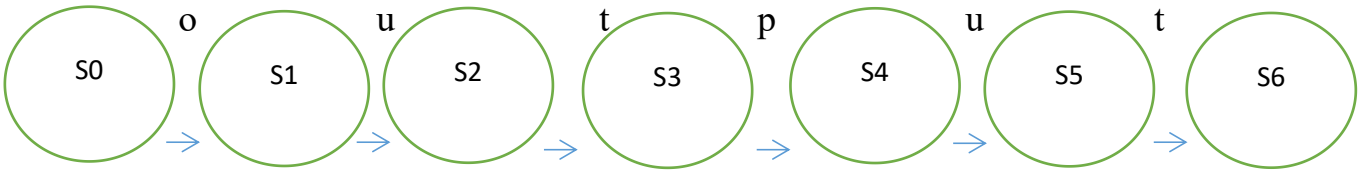

Рисунок 3.3 Пример графа переходов для цепочки output

3.10 Контрольный пример

Результат работы лексического анализатора в виде таблиц лексем и идентификаторов, соответствующих контрольному примеру, представлен в приложении Г.

4 Разработка синтаксического анализатора

4.1 Структура синтаксического анализатора

Синтаксический анализатор: часть компилятора, выполняющая синтаксический анализ, то есть исходный код проверяется на соответствие правилам грамматики. Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов. Выходной информацией– дерево разбора

Описание структуры синтаксического анализатора языка представлено на рисунке 4.1.

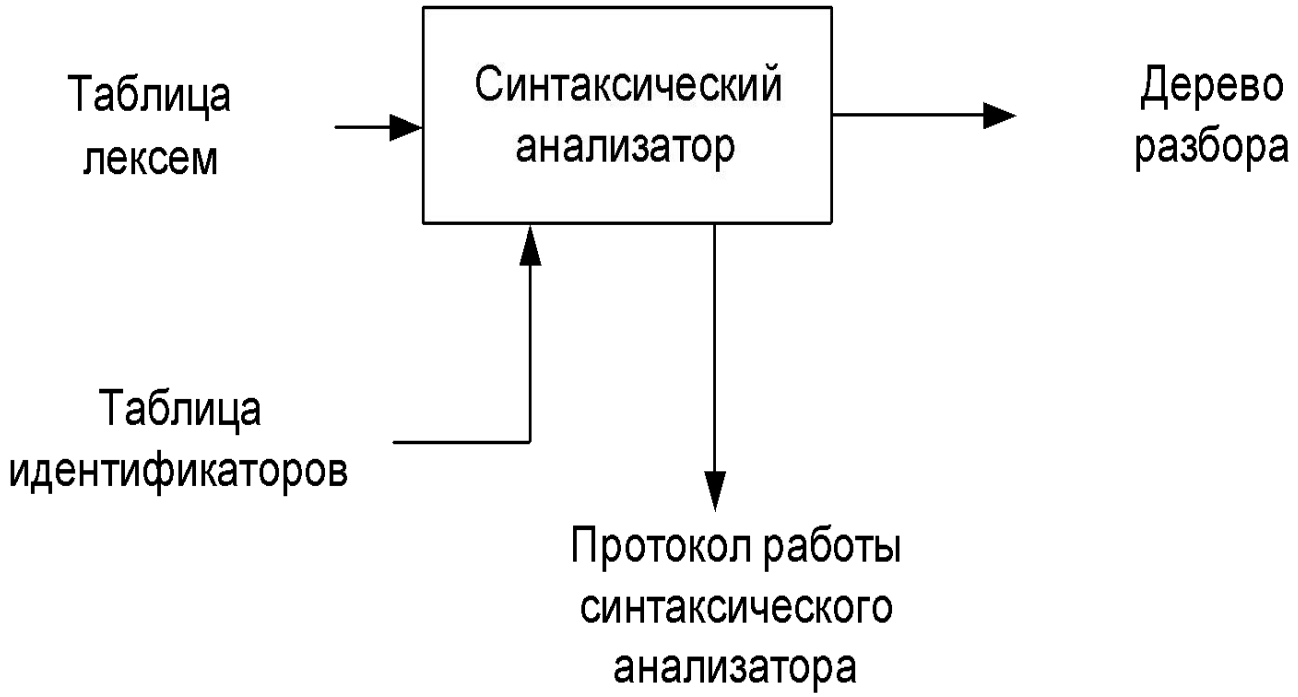

Рисунок 4.1 Структура синтаксического анализатора.

4.2 Контекстно-свободная грамматика, описывающая синтаксис языка

Грамматика, описывающая язык MMV-2020 представлена в таблице 4.1.

Таблица 4.1 грамматика языка MMV-2020

Нетереминалы	Описание
1	2

1	2
S	Правила, описывающие общую структуру программы
F	Порождает правила, описывающие формальные параметры функции
P	Порождает правила, описывающие формальные параметры функции при её объявлении
B	Порождает правила, описывающие тело функции
I	Порождает правила, описывающие литералы и идентификаторы
N	Порождает правила, описывающие инструкции языка
R	Порождает правила, описывающие логические действия
K	Порождает правила, описывающие вызов функции
E	Порождает правила, описывающие выражения
M	Порождает правила, описывающие арифметические действия
X	Порождает правила, описывающие конструкция в теле условного выражения

4.3 Построение конечного магазинного автомата

Конечный автомат с магазинной памятью представляет собой семерку $M = \langle Q, V, Z, \delta, q_0, z_0, F \rangle$. Подробное описание компонентов магазинного автомата представлено в таблице 4.2.

Таблица 4.2 Описание компонентов магазинного автомата

Компонента	Определение	Описание
1	2	3
Q	Множество состояний автомата	Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата
V	Алфавит входных символов	Алфавит представляет из себя множества терминальных и нетерминальных символов, описание которых содержится в таблица 3.1 и 4.1.
Z	Алфавит специальных магазинных символов	Алфавит магазинных символов содержит стартовый символ и маркер дна стека (представляет из себя символ $)
δ	Функция переходов автомата	Функция представляет из себя множество правил грамматики, описанных в таблице 4.1.
q_0	Начальное состояние автомата	Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики
z_0	Начальное состояние магазина автомата	Символ маркера дна стека $

1	2	3
F	Множество конечных состояний	Конечные состояние заставляют автомат прекратить свою работу. Конечным состоянием является пустой магазин автомата и совпадение позиции на входной ленте автомата с размером ленты

4.4 Основные структуры данных

Основные структуры данных синтаксического анализатора представляются в виде структуры магазинного конечного автомата, выполняющего разбор исходной ленты, и структуры грамматики Грейбах, описывающей синтаксические правила языка MMV-2020. Данные структуры в приложении Д.

4.5 Описание алгоритма синтаксического разбора

Принцип работы автомата следующий:

1. В магазин записывается стартовый символ;
2. На основе полученных ранее таблиц формируется входная лента;
3. Запускается автомат;
4. Выбирается цепочка, соответствующая нетерминальному символу, записывается в магазин в обратном порядке;
5. Если терминалы в стеке и в ленте совпадают, то данный терминал удаляется из ленты и стека. Иначе возвращаемся в предыдущее сохраненное состояние и выбираем другую цепочку нетерминала;
6. Если в магазине встретился нетерминал, переходим к пункту 4;
7. Если наш символ достиг дна стека, и лента в этот момент пуста, то синтаксический анализ выполнен успешно. Иначе генерируется исключение.

4.6 Структура и перечень сообщений синтаксического анализатора

Перечень сообщений, формируемых синтаксическим анализатором в ходе своей работы, представлен в таблице 4.3.

Таблица 4.3 Перечень сообщений лексического анализатора

Код ошибки	Сообщение
600	Неверная структура программы
601	Отсутствует список параметров функции при её объявлении
602	Ошибка в параметрах функции при её объявлении
603	Возможно отсутствует тело функции
604	Недопустимое выражение. Ожидаются только литералы и идентификаторы
605	Неверная конструкция в теле функции
606	Ошибка в условном выражении
607	Ошибка в вызове функции
608	Ошибка в арифметическом выражении
609	Ошибка в списке параметров при вызове функции
610	Неверная конструкция в условном выражении
611	Требуется закрывающаяся фигурная скобка
612	Требуется открывающаяся фигурная скобка

4.7. Параметры синтаксического анализатора и режимы его работы

Входной информацией для синтаксического анализатора является таблица лексем и идентификаторов. Кроме того, используется описание грамматики в форме Грейбах. Результаты работы лексического разбора, а именно дерево разбора

и протокол работы автомата с магазинной памятью выводятся в журнал работы программы.

4.8. Принцип обработки ошибок

Синтаксический анализатор выполняет разбор исходной последовательности лексем до тех пор, пока не дойдёт до конца цепочки лексем или не найдёт ошибку. Тогда анализ останавливается и выводится сообщение об ошибке (если она найдена).

4.9. Контрольный пример

Результаты работы лексического разбора, а именно дерево разбора и протокол работы автомата с магазинной памятью приведены в приложении Е.

5 Разработка семантического анализатора

5.1 Структура семантического анализатора

Семантический анализатор состоит из набора функций для проверки правильности исходной программы. Функции анализатора выполняются на различных этапах работы транслятора. Структура семантического анализатора представлена на рисунке 5.1.

Рисунок 5.1. Структура семантического анализатора

5.2 Функции семантического анализатора

Семантический анализатор выполняет проверку на основные правила языка (семантики языка), которые описаны в разделе 1.16.

5.3 Структура и перечень сообщений семантического анализатора

Перечень сообщений, формируемых семантического анализатором в ходе своей работы, представлен в таблице 5.1.

Таблица 5.1 Перечень сообщений семантического анализатора

Код ошибки	Сообщение
1	2
300	Не закрыт строковый литерал
301	Попытка использовать необьявленный идентификатор

1	2
302	Отсутствует точка входа head
303	Обнаружено несколько точек входа head
304	Превышен размер строкового литерала
305	Объявление переменной без ключевого слова var недопустимо
306	Необъявленный идентификатор
307	Недопустимо объявление переменной без указания типа
308	Попытка реализовать уже существующую функцию
309	Попытка переопределить параметр или ошибка области видимости
310	Попытка переопределить переменную или ошибка области видимости
311	Не указан тип функции
312	Использование пустого строкового литерала недопустимо
313	Тип функции и тип возвращаемого значения должны совпадать
314	Превышено максимально допустимое (2) количество параметров функции
315	Несовпадение типов передаваемых параметров
316	Слишком много аргументов в вызове функции
317	Слишком мало аргументов в вызове функции
318	Несовместимые типы при присваивании

1	2
319	Недопустимый целочисленный литерал
320	Типы данных в выражении не совпадают
321	Арифметические операторы могут применяться только к целочисленными типам
322	Логические операторы могут применяться только к целочисленному и логическому типами
323	К строкам можно применять только оператор +
324	К bool можно применять только логическое равно/неравно

5.4 Принцип обработки ошибок

Ошибки, возникающие в процессе трансляции программы, фиксируются в протокол, заданный входным параметрами. В случае возникновения ошибки происходит протоколирование с номером ошибки и диагностическим сообщением.

5.5 Контрольный пример

Соответствие примеров некоторых ошибок в исходном коде и диагностических сообщений об ошибках приведено в таблице 5.1.

Таблица 5.1 Примеры диагностики ошибок

Исходный код	Текст сообщения
1	2
head { int x = 9.	Error305 Семантическая ошибка -> Объявление переменной без ключевого слова var недопустимо Строка: 20

1	2
Output[x]. }	
head { var int x = 9. var str y = x. {	Error320 Семантическая ошибка -> Типы данных в выражении не совпадают Строка: 11

head
 	
{
 	
var int num = byte100.
 	
}
 	
head
 	
{
 	
var str.
 	
}
	Error303 Семантическая ошибка -> Обнаружено несколько точек входа Head

6 Вычисление выражений

6.1 Выражения, допускаемые языком

В языке MMV-2020 допускаются вычисления выражений целочисленного типа данных с поддержкой вызова функций внутри выражений. Приоритет операций представлен на таблице 6.1.

Таблица 6.1 Приоритеты операций

Операция	Значение приоритета
( )	3
*	2
%	2
/	2
+	1
-	1

6.2 Польская запись и принцип её построения

Язык программирования MMV-2020 не осуществляет преобразование выражений в польскую запись, поскольку в процессе генерации кода в этом не возникает необходимости.

7 Генерация кода

7.1 Структура генератора кода

Интерпретация кода – это часть процесса трансляции. Транслятор интерпретирует код на языке MMV-2020 в код на языке C# на основе таблицы лексем и идентификаторов. Схематично интерпретация кода показана на рисунке 7.1. Интерпретатор начинает свою работу только в том случае если код на языке MMV-2020 прошёл предыдущие компоненты транслятора без ошибок.

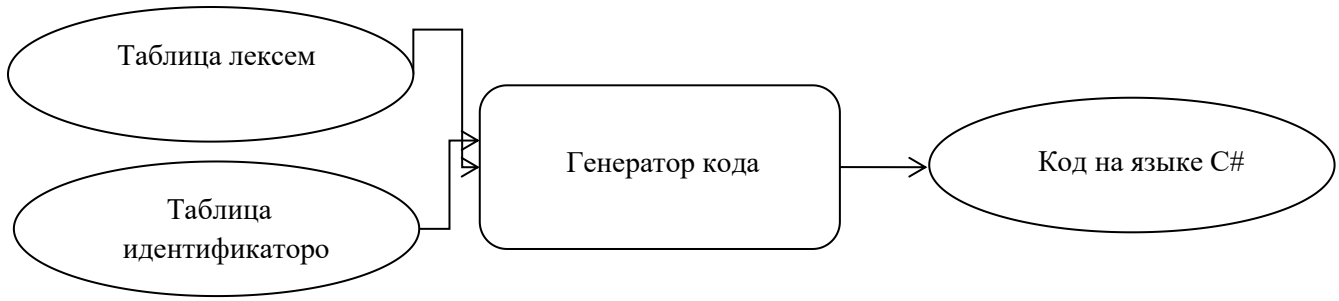

Рисунок 7.1 Структура генератора кода

7.2 Представление типов данных в оперативной памяти

При генерации кода на исходные типы языка MMV-2020 приводятся в соответствие к типам языка C#. В таблице 7.1 представлено соответствие типов данных исходного и получаемого в процессе генерации кода языков.

Таблица 7.1 Функции статической библиотеки

Тип данных на языке MMV-2020	Тип данных на языке C#
int(целочисленный один байт)	int
str(строковый)	string
bool(логический)	boolean

7.3 Статическая библиотека

Языком MMV-2020 предусмотрены функции стандартной библиотеки. Основной их целью является расширение возможностей реализации программного кода. Библиотека реализована в качестве статического класса на C#. Она по умолчанию подключается на этапе генерации кода к файлу, в который генерация осуществляется.

Таблица 7.2 Функции статической библиотеки

Функция	Назначение
int Pow()	Вывод на консоль строки str
int MaxNum()	Вывод на консоль целочисленной переменной num

7.4 Особенности алгоритма генерации кода

Главной особенностью генерации кода является то что полученный код на языке C# автоматически выполняется

7.5 Входные параметры генератора кода

На вход генератору кода поступают таблицы лексем и идентификаторов исходного код программы на языке MMV-20. Результаты работы генератора кода выводятся в файл с расширением .cs.

7.6 Контрольный пример

Результат генерации кода на основе контрольного примера из приложения А приведен в приложении Ж. Результат работы контрольного примера приведён на рисунке 7.2.

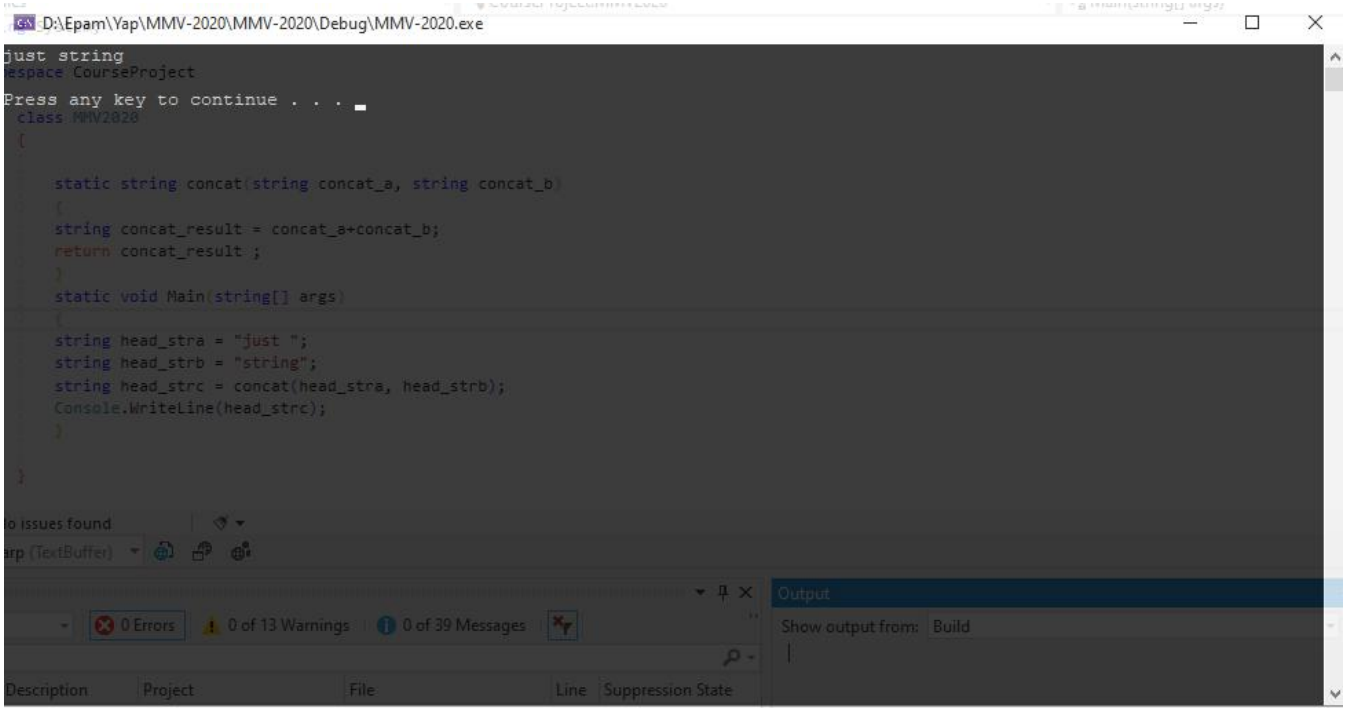

Рисунок 7.2 Результат работы программы на языке MMV-2020

8 Тестирование транслятора

8.1 Общие положения

Для демонстрации обработки ошибок выбраны наиболее наглядные примеры. Сообщение об ошибке, имеет уникальный номер, префикс, отвечающий за характер возникшей ошибки, а также краткое пояснение. В случаях, если неточность в коде формирует сразу несколько возможных ошибок, выводится сообщение о той, что была обнаружена первой.

8.2 Результаты тестирования

Таблица 8.1 Тестирование транслятора

Исходный код	Текст сообщения
1	2

head
{
int x = 9.
Output[x].
}
	Error305 Семантическая ошибка -> Объявление переменной без ключевого слова var недопустимо

head
{
int x = “string”.
Output[x].
}
	Error318 Семантическая ошибка -> Несовместимые типы при присваивании

function int value[int a]
{
Var str st = “string”.
return[st].
}
	Error313 Семантическая ошибка -> Тип функции и тип возвращаемого значения должны совпадать

head { var int x = 9. var str y = “string”. Pow[x & y]. {	Error315 Семантическая ошибка -> Несовпадение типов передаваемых параметров

1	2

head
{
var int x = 9.
var bool bl = true.
check[x < 9].
{
bl = false.
}
{
	Error322 Семантическая ошибка -> Логические операторы могут применяться только к целочисленному и логическому типами (типы должны совпадать)

head { var int x = 9. var str y = x. {	Error320 Семантическая ошибка -> Типы данных в выражении не совпадают Строка: 11

function value[int a]
{
Var str st = “string”.
return[st].
}
	Error311 Семантическая ошибка -> Не указан тип функции

head
{
var int lhhbj = byte100.
}
head
{
var str.
}
	Error303 Семантическая ошибка -> Обнаружено несколько точек входа head

Заключение

В ходе выполнения курсовой работы был разработан транслятор и генератор кода для языка программирования MMV-2020 со всеми необходимыми компонентами. Таким образом, были выполнены основные задачи данной курсовой работы:

1. Сформулирована спецификация языка MMV-2020;
2. Разработаны конечные автоматы и важные алгоритмы на их основе для эффективной работы лексического анализатора;
3. Осуществлена программная реализация лексического анализатора, распознающего допустимые цепочки спроектированного языка;
4. Разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
5. Осуществлена программная реализация синтаксического анализатора;
6. Разработан семантический анализатор, осуществляющий проверку используемых инструкций на соответствие логическим правилам;
7. Разработан транслятор кода на язык C#;
8. Проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка MMV-2020 включает:

1. 3 типа данных;
2. Возможность вызова функций стандартной библиотеки;
3. Наличие 5 арифметических операторов для вычисления выражений;
4. Наличие 6 логических операторов
5. Поддержка функций, оператора условия;
6. Структурированная и классифицированная система для обработки ошибок пользователя.

Проделанная работа позволила получить необходимое представление о структурах и процессах, использующихся при построении трансляторов, а также основные различия и преимущества тех или иных средств трансляции.

ПРИЛОЖЕНИЕ А – ЛИСТИНГ ИСХОДНОГО КОДА

Файлы:

1) example.txt

```
function int math [int one & int two]
{
	var int result.
	result = (one + Pow[one & two ]) % two .
	return[result].
}

function str concat[str a & str b]
{
	var str result = a + b.
	return[result].
}

function int ten[]
{
	return[10].
}

head
{
	var int l = byte100.
	output [l].

	var int num = 010.
	output [num].

	var bool bl = true.
	output [bl].

	var int fdfshhvhbhj.
	var int c= ten[].
	output [c].

	var int s=octal4.
	output [s].

	var int f = math[l & s].
	output[f].
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А

```
  var str stra = "just ".
 var str strb = "string".
 var str strc = concat[stra & strb].
 output [strc].

  var int x = hex24.
  var int rc = MaxNum[x & s].
 output [rc].

  check [rc > f]
  {
      output[x].
  }
  otherwise
  {
          output[strc].
  }
}
```

Рис. 8 - Исходный код

ПРИЛОЖЕНИЕ Б – ЛИСТИНГ ПРОГРАММЫ

Файлы:

1) PredefinedWords.h

```
#define EXPRESSION_INT 4, \
        FST::NODE(1,FST::RELATION('i',1)),\
        FST::NODE(1,FST::RELATION('n',2)),\
        FST::NODE(1,FST::RELATION('t',3)),\
        FST::NODE()

#define EXPRESSION_STR 4, \
        FST::NODE(1,FST::RELATION('s',1)),\
        FST::NODE(1,FST::RELATION('t',2)),\
        FST::NODE(1,FST::RELATION('r',3)),\
        FST::NODE()

#define EXPRESSION_NEWSTR 8,\
        FST::NODE(1, FST::RELATION('N', 1)), \
        FST::NODE(1, FST::RELATION('e', 2)), \
        FST::NODE(1, FST::RELATION('w', 3)), \
        FST::NODE(1, FST::RELATION('L', 4)), \
    FST::NODE(1, FST::RELATION('i', 5)), \
        FST::NODE(1, FST::RELATION('n', 6)), \
        FST::NODE(1, FST::RELATION('e', 7)), \
        FST::NODE()

#define EXPRESSION_ARITHMETIC 2,   \
        FST::NODE(5,  FST::RELATION('+',  1),  FST::RELATION('-',  1),
FST::RELATION('*', 1), FST::RELATION('/', 1), FST::RELATION('%', 1)),\
        FST::NODE()

#define EXPRESSION_LOG_OPERATOR 2, \
        FST::NODE(6,  FST::RELATION('>',  1),  FST::RELATION('<',  1),
FST::RELATION(`',1),  FST::RELATION('~',  1),  FST::RELATION('?',  1),
FST::RELATION('#', 1)), \
        FST::NODE()


#define EXPRESSION_POINT 2,     \
        FST::NODE(1, FST::RELATION('.', 1)),\
        FST::NODE()

#define EXPRESSION_AND 2,       \
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б

```
        FST::NODE(1, FST::RELATION('&', 1)),\
        FST::NODE()

#define EXPRESSION_LEFTBRACE 2,     \
        FST::NODE(1, FST::RELATION('{', 1)),\
        FST::NODE()

#define EXPRESSION_BRACELET 2,      \
        FST::NODE(1, FST::RELATION('}', 1)),\
        FST::NODE()

#define EXPRESSION_LEFTTHESIS 2,    \
        FST::NODE(1, FST::RELATION('[', 1)),\
        FST::NODE()

#define EXPRESSION_RIGHTTHESIS 2,   \
        FST::NODE(1, FST::RELATION(']', 1)),\
        FST::NODE()

#define EXPRESSION_LEFTBRACKET 2,       \
        FST::NODE(1, FST::RELATION('(', 1)),\
        FST::NODE()

#define EXPRESSION_RIGHTBRACKET 2,      \
        FST::NODE(1, FST::RELATION(')', 1)),\
        FST::NODE()

#define EXPRESSION_EQUAL 2,     \
        FST::NODE(1, FST::RELATION('=', 1)),\
        FST::NODE()

#define EXPRESSION_SEPARATOR 2,     \
        FST::NODE(1, FST::RELATION('|', 1)),\
        FST::NODE()

#define EXPRESSION_ID 2, \
        FST::NODE(52,    \
        FST::RELATION('a', 0), FST::RELATION('b', 0), FST::RELATION('c', 0), FST::RELATION('d', 0), FST::RELATION('e', 0), FST::RELATION('f', 0),\
        FST::RELATION('g', 0), FST::RELATION('h', 0), FST::RELATION('i', 0), FST::RELATION('j', 0), FST::RELATION('k', 0), FST::RELATION('l', 0),\
        FST::RELATION('m', 0), FST::RELATION('n', 0), FST::RELATION('o', 0), FST::RELATION('p', 0), FST::RELATION('q', 0), FST::RELATION('r', 0),\
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б

```
        FST::RELATION('s', 0), FST::RELATION('t', 0), FST::RELATION('u', 0), FST::RELATION('v', 0), FST::RELATION('w', 0), FST::RELATION('x', 0),\
        FST::RELATION('y', 0), FST::RELATION('z', 0),\
        \
        FST::RELATION('a', 1), FST::RELATION('b', 1), FST::RELATION('c', 1), FST::RELATION('d', 1), FST::RELATION('e', 1), FST::RELATION('f', 1),\
        FST::RELATION('g', 1), FST::RELATION('h', 1), FST::RELATION('i', 1), FST::RELATION('j', 1), FST::RELATION('k', 1), FST::RELATION('l', 1),\
        FST::RELATION('m', 1), FST::RELATION('n', 1), FST::RELATION('o', 1), FST::RELATION('p', 1), FST::RELATION('q', 1), FST::RELATION('r', 1),\
        FST::RELATION('s', 1), FST::RELATION('t', 1), FST::RELATION('u', 1), FST::RELATION('v', 1), FST::RELATION('w', 1), FST::RELATION('x', 1),\
        FST::RELATION('y', 1), FST::RELATION('z', 1)),\
        FST::NODE()

#define EXPRESSION_STRING_LITERAL 3,\
        FST::NODE(1, FST::RELATION('\'', 1)),\
        FST::NODE(89, \
                        FST::RELATION('a', 1), FST::RELATION('b', 1), FST::RELATION('c', 1), FST::RELATION('d', 1),\
                        FST::RELATION('e', 1), FST::RELATION('f', 1), FST::RELATION('g', 1), FST::RELATION('h', 1),\
                        FST::RELATION('i', 1), FST::RELATION('j', 1), FST::RELATION('k', 1), FST::RELATION('l', 1),\
                        FST::RELATION('m', 1), FST::RELATION('n', 1), FST::RELATION('o', 1), FST::RELATION('p', 1),\
                        FST::RELATION('q', 1), FST::RELATION('r', 1), FST::RELATION('s', 1), FST::RELATION('t', 1),\
                        FST::RELATION('u', 1), FST::RELATION('v', 1), FST::RELATION('w', 1), FST::RELATION('x', 1),\
                        FST::RELATION('y', 1), FST::RELATION('z', 1),\
                        FST::RELATION('A', 1), FST::RELATION('B', 1), FST::RELATION('C', 1), FST::RELATION('D', 1),\
                        FST::RELATION('E', 1), FST::RELATION('F', 1), FST::RELATION('G', 1), FST::RELATION('H', 1),\
                        FST::RELATION('I', 1), FST::RELATION('J', 1), FST::RELATION('K', 1), FST::RELATION('L', 1),\
                        FST::RELATION('M', 1), FST::RELATION('N', 1), FST::RELATION('O', 1), FST::RELATION('P', 1),\
                        FST::RELATION('Q', 1), FST::RELATION('R', 1), FST::RELATION('S', 1), FST::RELATION('T', 1),\
                        FST::RELATION('U', 1), FST::RELATION('V', 1), FST::RELATION('W', 1), FST::RELATION('X', 1),\
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б

```
                FST::RELATION('Y', 1), FST::RELATION('Z', 1),\
                FST::RELATION('0', 1), FST::RELATION('1', 1),
FST::RELATION('2', 1), FST::RELATION('3', 1),\
                FST::RELATION('4', 1), FST::RELATION('5', 1),
FST::RELATION('6', 1), FST::RELATION('7', 1),\
                FST::RELATION('8', 1), FST::RELATION('9', 1),\
                FST::RELATION(' ', 1), FST::RELATION(',', 1),
FST::RELATION('.', 1), FST::RELATION(';', 1),\
                FST::RELATION('-', 1), FST::RELATION('+', 1),
FST::RELATION('*', 1), FST::RELATION('/', 1),\
                FST::RELATION('=', 1), FST::RELATION(':', 1),
FST::RELATION(')', 1), FST::RELATION('(', 1),\
                FST::RELATION('}', 1), FST::RELATION('{', 1),
FST::RELATION(']', 1), FST::RELATION('[', 1),\
                FST::RELATION('!', 1), FST::RELATION('?', 1),
FST::RELATION('#', 1), FST::RELATION('&', 1),\
                FST::RELATION('>', 1), FST::RELATION('<', 1),
FST::RELATION('<=',1), FST::RELATION('>=',1),\
                FST::RELATION('[', 1), FST::RELATION(']', 1),
FST::RELATION('\'', 2)),\
        FST::NODE()

#define EXPRESSION_INT_LITERAL 3, \
        FST::NODE(11, \
                FST::RELATION('-', 1),\
                FST::RELATION('0',2),FST::RELATION('1',2),\
                FST::RELATION('2',2),FST::RELATION('3',2),\
                FST::RELATION('4',2),FST::RELATION('5',2),\
                FST::RELATION('6',2),FST::RELATION('7',2),\
                FST::RELATION('8',2),FST::RELATION('9',2)),\
        FST::NODE(9,\
                FST::RELATION('1',2),\
                FST::RELATION('2',2),FST::RELATION('3',2),\
                FST::RELATION('4',2),FST::RELATION('5',2),\
                FST::RELATION('6',2),FST::RELATION('7',2),\
                FST::RELATION('8',2),FST::RELATION('9',2)),\
        FST::NODE(10,\
                FST::RELATION('0',2),FST::RELATION('1',2),\
                FST::RELATION('2',2),FST::RELATION('3',2),\
                FST::RELATION('4',2),FST::RELATION('5',2),\
                FST::RELATION('6',2),FST::RELATION('7',2),\
                FST::RELATION('8',2),FST::RELATION('9',2))
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б

```
#define EXPRESSION_VAR 4, \
        FST::NODE(1,FST::RELATION('v',1)),\
        FST::NODE(1,FST::RELATION('a',2)),\
        FST::NODE(1,FST::RELATION('r',3)),\
        FST::NODE()

#define EXPRESSION_FUNCTION 9, \
        FST::NODE(1, FST::RELATION('f', 1)),\
    FST::NODE(1, FST::RELATION('u', 2)),\
    FST::NODE(1, FST::RELATION('n', 3)),\
    FST::NODE(1, FST::RELATION('c', 4)),\
    FST::NODE(1, FST::RELATION('t', 5)),\
    FST::NODE(1, FST::RELATION('i', 6)),\
    FST::NODE(1, FST::RELATION('o', 7)),\
    FST::NODE(1, FST::RELATION('n', 8)),\
    FST::NODE()

#define EXPRESSION_POW 4, \
        FST::NODE(1, FST::RELATION('P', 1)),\
    FST::NODE(1, FST::RELATION('o', 2)),\
    FST::NODE(1, FST::RELATION('w', 3)),\
    FST::NODE()
#define EXPRESSION_MAXNUM 7,\
        FST::NODE(1, FST::RELATION('M', 1)),\
    FST::NODE(1, FST::RELATION('a', 2)),\
    FST::NODE(1, FST::RELATION('x', 3)),\
        FST::NODE(1, FST::RELATION('N', 4)),\
    FST::NODE(1, FST::RELATION('u', 5)),\
        FST::NODE(1, FST::RELATION('m', 6)),\
    FST::NODE()

#define EXPRESSION_HEAD 5, \
        FST::NODE(1,FST::RELATION('h',1)),\
        FST::NODE(1,FST::RELATION('e',2)),\
        FST::NODE(1,FST::RELATION('a',3)),\
        FST::NODE(1,FST::RELATION('d',4)),\
        FST::NODE()

#define EXPRESSION_CHECK 6, \
        FST::NODE(1,FST::RELATION('c',1)),\
        FST::NODE(1,FST::RELATION('h',2)),\
        FST::NODE(1,FST::RELATION('e',3)),\
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б

```
        FST::NODE(1,FST::RELATION('c',4)),\

        FST::NODE(1,FST::RELATION('k',5)),\
        FST::NODE()

#define EXPRESSION_OTHERWISE 10, \
        FST::NODE(1,FST::RELATION('o',1)),\
        FST::NODE(1,FST::RELATION('t',2)),\
        FST::NODE(1,FST::RELATION('h',3)),\
        FST::NODE(1,FST::RELATION('e',4)),\
        FST::NODE(1,FST::RELATION('r',5)),\
        FST::NODE(1,FST::RELATION('w',6)),\
        FST::NODE(1,FST::RELATION('i',7)),\
        FST::NODE(1,FST::RELATION('s',8)),\
        FST::NODE(1,FST::RELATION('e',9)),\
        FST::NODE()

#define EXPRESSION_OUT 4, \
        FST::NODE(1,FST::RELATION('o',1)),\
        FST::NODE(1,FST::RELATION('u',2)),\
        FST::NODE(1,FST::RELATION('t',3)),\
    FST::NODE(1,FST::RELATION('p',1)),\
        FST::NODE(1,FST::RELATION('u',2)),\
        FST::NODE(1,FST::RELATION('t',3)),\
        FST::NODE()

#define EXPRESSION_RETURN 7, \
        FST::NODE(1,FST::RELATION('r',1)),\
        FST::NODE(1,FST::RELATION('e',2)),\
        FST::NODE(1,FST::RELATION('t',3)),\
        FST::NODE(1,FST::RELATION('u',4)),\
        FST::NODE(1,FST::RELATION('r',5)),\
        FST::NODE(1,FST::RELATION('n',6)),\
        FST::NODE()

#define EXPRESSION_TRUE 5, \
        FST::NODE(1,FST::RELATION('t',1)),\
        FST::NODE(1,FST::RELATION('r',2)),\
        FST::NODE(1,FST::RELATION('u',3)),\
        FST::NODE(1,FST::RELATION('e',4)),\
        FST::NODE()

#define EXPRESSION_FALSE 6, \
        FST::NODE(1,FST::RELATION('f',1)),\
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б

```
        FST::NODE(1,FST::RELATION('a',2)),\

        FST::NODE(1,FST::RELATION('l',3)),\
        FST::NODE(1,FST::RELATION('s',4)),\
        FST::NODE(1,FST::RELATION('e',5)),\
        FST::NODE()

#define EXPRESSION_BOOL 5, \
        FST::NODE(1,FST::RELATION('b',1)),\
        FST::NODE(1,FST::RELATION('o',2)),\
        FST::NODE(1,FST::RELATION('o',3)),\
        FST::NODE(1,FST::RELATION('l',4)),\
        FST::NODE()

#define EXPRESSION_INT8 7, \
        FST::NODE(1, FST::RELATION('o', 1)), \
    FST::NODE(1, FST::RELATION('c', 2)), \
    FST::NODE(1, FST::RELATION('t', 3)), \
    FST::NODE(1, FST::RELATION('a', 4)), \
    FST::NODE(1, FST::RELATION('l', 5)), \
        FST::NODE(16, FST::RELATION('0', 5), FST::RELATION('1', 5), FST::RELATION('2', 5), FST::RELATION('3', 5),\
        FST::RELATION('4', 5), FST::RELATION('5', 5), FST::RELATION('6', 5), FST::RELATION('7', 5),\
        FST::RELATION('0', 6), FST::RELATION('1', 6), FST::RELATION('2', 6), FST::RELATION('3', 6),\
        FST::RELATION('4', 6), FST::RELATION('5', 6), FST::RELATION('6', 6), FST::RELATION('7', 6)),\
        FST::NODE()

#define EXPRESSION_INT2 6, \
FST::NODE(1, FST::RELATION('b', 1)), \
FST::NODE(1, FST::RELATION('y', 2)), \
FST::NODE(1, FST::RELATION('t', 3)), \
FST::NODE(1, FST::RELATION('e', 4)), \
FST::NODE(4, FST::RELATION('0', 4), FST::RELATION('1', 4), FST::RELATION('0', 5), FST::RELATION('1', 5)), \
FST::NODE()

#define EXPRESSION_INT16 5,\
FST::NODE(1, FST::RELATION('h', 1)), \
         FST::NODE(1, FST::RELATION('e', 2)), \
    FST::NODE(1, FST::RELATION('x', 3)), \
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Б

```
FST::NODE(32, FST::RELATION('0', 3), FST::RELATION('1', 3), FST::RELATION('2', 3), FST::RELATION('3', 3), FST::RELATION('4', 3), FST::RELATION('5', 3),\
    FST::RELATION('6', 3), FST::RELATION('7', 3), FST::RELATION('8', 3), FST::RELATION('9', 3), \
        FST::RELATION('a', 3), FST::RELATION('b', 3), FST::RELATION('c', 3), FST::RELATION('d', 3), FST::RELATION('e', 3), FST::RELATION('f', 4), \
        FST::RELATION('0', 4), FST::RELATION('1', 4), FST::RELATION('2', 4), FST::RELATION('3', 4), FST::RELATION('4', 4), FST::RELATION('5', 4),\
    FST::RELATION('6', 4), FST::RELATION('7', 4), FST::RELATION('8', 4), FST::RELATION('9', 4), \
        FST::RELATION('a', 4), FST::RELATION('b', 4), FST::RELATION('c', 4), FST::RELATION('d', 4), FST::RELATION('e', 4), FST::RELATION('f', 4)), \
    FST::NODE()
```

ПРИЛОЖЕНИЕ В – ЛИСТИНГ ПРОГРАММЫ

Файлы заголовков:

2) IT.h – структуры данных для таблицы идентификаторов

```
enum IDDATATYPE { INT = 1, STR = 2, BOOL = 3, UNKNOWN };
	// типы данных идентификаторов: integer, string, bool, неопределенный

enum IDTYPE { V = 1, F = 2, P = 3, L = 4, S = 5 };
	// типы идентификаторов: переменная, функция, параметр, литерал, библ. функция

static const IDDATATYPE POW_PARAMS[] = { IT::IDDATATYPE::INT, IT::IDDATATYPE::INT };

static const IDDATATYPE MAXNUM_PARAMS[] = { IT::IDDATATYPE::INT, IT::IDDATATYPE::INT };


struct Entry

{

	int 			idxfirstLE;

	unsigned char id[ID_MAXSIZE];

	IDDATATYPE	iddatatype = IDDATATYPE::UNKNOWN;

	IDTYPE		idtype;

	unsigned char visibility[ID_MAXSIZE];

	struct

	{

		int vint;

		double vdec;

		struct

		{

			int len;

			unsigned char str[TI_STR_MAXSIZE - 1];

		} vstr;

		struct

		{

			int count;

			IDDATATYPE *types;
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ В

```
            } params;
        } value = { LT_TI_NULLIDX };

};
struct IdTable
{
        int maxsize;
        int size;
        Entry* table;
};
```

3) LT.h – структуры данных для таблицы лексем

```
enum OPER { NOT = -1, PLUS = 1, MINUS, MUL, DIR, OST, MORE, LESS, LOREQU, MOREQU, NEQU, EQU,
NOEQU };
struct Entry                        // строка таблицы лексем
{
        unsigned char lexema;       // лексема
        int line;                   // номер строки в исходном тексте
         int idxTI;                 // индекс в таблице идентификаторов или LT_TI_NULLIDX
        int priority;               // приоритет
        unsigned char sType;
         OPER operation;
        bool polish;
};
struct LexTable                     // экземпляр таблицы лексем
{
        int maxsize;                // емкость таблицы лексем < LT_MAXSIZE
        int size;                   // текущий размер таблицы лексем < maxsize
        Entry* table;               // массив строк таблицы лексем
};
```

ПРИЛОЖЕНИЕ Г

1) Таблица лексем

```
1 fti[ti&ti]
2 {
3 vti.
4 i=(isP[i&i])si.
5 r[i].
6 }
8 fti[ti&ti]
9 {
10 vti=isi.
11 r[i].
12 }
13 fti[]
14 {
15 r[l].
16 }
18 h
19 {
20 vti=l.
21 o[i].
23 vti=l.
24 o[i].
26 vti=l.
27 o[i].
29 vti.
30 vti=i[].
31 o[i].
33 vti=l.
34 o[i].
36 vti=i[i&i].
37 o[i].
39 vti=l.
40 vti=l.
41 vti=i[i&i].
42 o[i].
44 vti=l.
45 vti=S[i&i].
46 o[i].
48 c[ibi]
49 {
50 o[i].
51 }
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Г

```
52 u
53 {
54 o[i].
55 }
56 }
```

2) Таблица идентификаторов

Таблица идентификаторов

№	Идентификатор	Тип данных	Тип идентификатора	Индекс в ТЛ	Значение
0	math	integer	функция	2	1:integer 2:integer
1	math_one	integer	параметр	5	-
2	math_two	integer	параметр	8	-
3	math_result	integer	переменная	13	0
4	Pow	integer	библ. функция	20	1:integer 2:integer
5	concat	string	функция	38	1:string 2:string
6	concat_a	string	параметр	41	-
7	concat_b	string	параметр	44	-
8	concat_result	string	переменная	49	[0]""
9	ten	integer	функция	63	-
10	L1	integer	литерал	69	10
11	head_l	integer	переменная	77	0
12	L2	integer	литерал	79	4
13	head_num	integer	переменная	88	0
14	head_bl	bool	переменная	99	0
15	L3	bool	литерал	101	1
16	head_fdfshhvh	integer	переменная	110	0
17	head_c	integer	переменная	114	0
18	head_s	integer	переменная	127	0
19	head_f	integer	переменная	138	0
20	head_stra	string	переменная	154	[0]""
21	L4	string	литерал	156	[5]"just "
22	head_strb	string	переменная	160	[0]""
23	L5	string	литерал	162	[6]"string"
24	head_strc	string	переменная	166	[0]""
25	head_x	integer	переменная	182	0
26	L6	integer	литерал	184	36
27	head_rc	integer	переменная	188	0
28	MaxNum	integer	библ. функция	190	1:integer 2:integer

Рисунок 1 Таблица идентификаторов

ПРИЛОЖЕНИЕ Д – ЛИСТИНГ ПРОГРАММЫ

Файлы заголовков:

1) Rule.h

```
Greibach greibach(
              NS('S'), TS('$'),                    // стартовый символ, дно стека
              13,                                  // количество правил
       Rule(
              NS('S'), GRB_ERROR_SERIES + 0,    // неверная структура программы
              3,
              Rule::Chain(6, TS('f'), TS('t'), TS('i'), NS('F'), NS('B'), NS('S')),
              Rule::Chain(4, TS('h'), TS('{'), NS('N'), TS('}')),
              Rule::Chain(5, TS('f'), TS('t'), TS('i'), NS('F'), NS('B'))
       ),
       Rule(
              NS('F'), GRB_ERROR_SERIES + 1,    // Отсутствует список параметров функции
              2,
              Rule::Chain(3, TS('['), NS('P'), TS(']')),
              Rule::Chain(2, TS('['), TS(']'))         ),
       Rule(
              NS('P'), GRB_ERROR_SERIES + 2,    // Ошибка в параметрах функции при её объявлении
              2,
              Rule::Chain(2, TS('t'), TS('i')),
              Rule::Chain(4, TS('t'), TS('i'), TS('&'), NS('P'))
       ),
       Rule(
              NS('B'), GRB_ERROR_SERIES + 3,    // Отсутствует тело функции
              2,
              Rule::Chain(8, TS('{'), NS('N'), TS('r'), TS('['), NS('I'), TS(']'), TS('.'),
TS('}')),
              Rule::Chain(7, TS('{'), TS('r'), TS('['), NS('I'), TS(']'), TS('.'), TS('}'))
       ),
       Rule(
              NS('I'), GRB_ERROR_SERIES + 4,    // Недопустимое выражение. Ожидаются только
литералы и идентификаторы
              2,
              Rule::Chain(1, TS('l')),
              Rule::Chain(1, TS('i'))
       ),
       Rule(
              NS('N'), GRB_ERROR_SERIES + 5,    // Неверная конструкция в теле функции
              20,
              Rule::Chain(5, TS('v'), TS('t'), TS('i'), TS('.'), NS('N')),//
              Rule::Chain(7, TS('v'), TS('t'), TS('i'), TS('='), NS('E'), TS('.'), NS('N')),//
              Rule::Chain(5, TS('i'), TS('='), NS('E'), TS('.'), NS('N')),//
              Rule::Chain(8, TS('c'), TS('['), NS('R'), TS(']'), TS('{'), NS('X'), TS('}'),
NS('N')),//if
              Rule::Chain(12, TS('c'), TS('['), NS('R'), TS(']'), TS('{'), NS('X'), TS('}'),
TS('u'), TS('{'), NS('X'), TS('}'), NS('N')),
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Д

```
            Rule::Chain(6, TS('P'), NS('K'), TS('.'), NS('N')),//
            Rule::Chain(6, TS('S'), NS('K'), TS('.'), NS('N')),//
            Rule::Chain(6, TS('o'), TS('['), NS('I'), TS(']'), TS('.'), NS('N')),//
            Rule::Chain(3, TS('n'), TS('.'), NS('N')),//
            Rule::Chain(4, TS('i'), NS('K'), TS('.'), NS('N')),

            Rule::Chain(4, TS('v'), TS('t'), TS('i'), TS('.')),
            Rule::Chain(6, TS('v'), TS('t'), TS('i'), TS('='), NS('E'), TS('.')),
            Rule::Chain(4, TS('i'), TS('='), NS('E'), TS('.')),
            Rule::Chain(7, TS('c'), TS('['), NS('R'), TS(']'), TS('{'), NS('X'), TS('}')),
            Rule::Chain(11, TS('c'), TS('['), NS('R'), TS(']'), TS('{'), NS('X'), TS('}'),
TS('u'), TS('{'), NS('X'), TS('}')),
            Rule::Chain(3, TS('P'), NS('K'), TS('.')),
            Rule::Chain(3, TS('S'), NS('K'), TS('.')),
            Rule::Chain(3, TS('o'), NS('K'), TS('.')),
            Rule::Chain(2, TS('n'), TS('.')),
            Rule::Chain(3, TS('i'), NS('K'), TS('.'))
      ),
      Rule(
            NS('R'), GRB_ERROR_SERIES + 6,    // Ошибка в условном выражении
            4,
            Rule::Chain(3, TS('i'), TS('b'), TS('i')),
            Rule::Chain(3, TS('i'), TS('b'), TS('l')),
            Rule::Chain(3, TS('l'), TS('b'), TS('i')),
            Rule::Chain(3, TS('l'), TS('b'), TS('l'))
      ),
      Rule(
            NS('K'), GRB_ERROR_SERIES + 7,    // Ошибка в вызове функции
            2,
            Rule::Chain(3, TS('['), NS('W'), TS(']')),
            Rule::Chain(2, TS('['), TS(']'))
      ),
      Rule(
            NS('E'), GRB_ERROR_SERIES + 8,    // ошибка в арифметическом выражении
            12,
            Rule::Chain(2, TS('i'), NS('K')),
            Rule::Chain(1, TS('i')),
            Rule::Chain(1, TS('l')),
            Rule::Chain(3, TS('('), NS('E'), TS(')')),
            Rule::Chain(2, TS('P'), NS('K')),
            Rule::Chain(2, TS('S'), NS('K')),

            Rule::Chain(2, TS('i'), NS('M')),
            Rule::Chain(2, TS('l'), NS('M')),
            Rule::Chain(4, TS('('), NS('E'), TS(')'), NS('M')),
            Rule::Chain(3, TS('i'), NS('K'), NS('M')),
            Rule::Chain(3, TS('P'), NS('K'), NS('M')),
            Rule::Chain(3, TS('S'), NS('K'), NS('M'))
      ),
      Rule(
            NS('W'), GRB_ERROR_SERIES + 9,    // ошибка в параметрах вызываемой функции
            4,
            Rule::Chain(1, TS('i')),
            Rule::Chain(1, TS('l')),
```

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Д

```
                Rule::Chain(3, TS('i'), TS('&'), NS('W')),
                Rule::Chain(3, TS('l'), TS('&'), NS('W'))
        ),
        Rule(
                NS('M'), GRB_ERROR_SERIES + 8,    // ошибка в арифметическом выражении
                2,
                Rule::Chain(2, TS('s'), NS('E')),
                Rule::Chain(3, TS('s'), NS('E'), NS('M'))
        ),
        Rule(
                NS('X'), GRB_ERROR_SERIES + 11,    // Неверная конструкция в теле цикла/условного
выражения
                12,
                Rule::Chain(5, TS('i'), TS('='), NS('E'), TS('.'), NS('X')),
                Rule::Chain(6, TS('P'), NS('K'), TS('.'), NS('X')),
                Rule::Chain(6, TS('S'), NS('K'), TS('.'), NS('X')),
                Rule::Chain(6, TS('o'), TS('['), NS('I'), TS(']'), TS('.'), NS('X')),
                Rule::Chain(3, TS('c'), TS('.'), NS('N')),
                Rule::Chain(4, TS('i'), NS('K'), TS('.'), NS('X')),

                Rule::Chain(4, TS('i'), TS('='), NS('E'), TS('.')),
                Rule::Chain(3, TS('P'), NS('K'), TS('.')),
                Rule::Chain(3, TS('S'), NS('K'), TS('.')),
                Rule::Chain(3, TS('o'), NS('K'), TS('.')),
                Rule::Chain(2, TS('c'), TS('.')),
                Rule::Chain(3, TS('i'), NS('K'), TS('.'))
        )
);
```

ПРИЛОЖЕНИЕ Е

СИНТАКСИЧЕСКИЙ АНАЛИЗ

Шаг	Правило	Входная лента	Стек
0000	S->ftiFBS	fti[ti&ti]{vti.i=(isP[i&i	S$
0000	SAVESTATE: 0001		
0000		fti[ti&ti]{vti.i=(isP[i&i	ftiFBS$
0001		ti[ti&ti]{vti.i=(isP[i&i]	tiFBS$
0002		i[ti&ti]{vti.i=(isP[i&i])	iFBS$
0003		[ti&ti]{vti.i=(isP[i&i])s	FBS$
0004	F->[P]	[ti&ti]{vti.i=(isP[i&i])s	FBS$
0004	SAVESTATE: 0002		
0004		[ti&ti]{vti.i=(isP[i&i])s	[P]BS$
0005		ti&ti]{vti.i=(isP[i&i])si	P]BS$
0006	P->ti	ti&ti]{vti.i=(isP[i&i])si	P]BS$
0006	SAVESTATE: 0003		
0006		ti&ti]{vti.i=(isP[i&i])si	ti]BS$
0007		i&ti]{vti.i=(isP[i&i])si.	i]BS$
0008		&ti]{vti.i=(isP[i&i])si.r	]BS$
0009	TS_NOK/NS_NORULECHAIN		
0009	RESSTATE		
0009		ti&ti]{vti.i=(isP[i&i])si	P]BS$
0010	P->ti&P	ti&ti]{vti.i=(isP[i&i])si	P]BS$
0010	SAVESTATE: 0003		
0010		ti&ti]{vti.i=(isP[i&i])si	ti&P]BS$
0011		i&ti]{vti.i=(isP[i&i])si.	i&P]BS$
0012		&ti]{vti.i=(isP[i&i])si.r	&P]BS$
0013		ti]{vti.i=(isP[i&i])si.r[	P]BS$
0014	P->ti	ti]{vti.i=(isP[i&i])si.r[	P]BS$
0014	SAVESTATE: 0004		
0014		ti]{vti.i=(isP[i&i])si.r[	ti]BS$
0015		i]{vti.i=(isP[i&i])si.r[i	i]BS$
0016		]{vti.i=(isP[i&i])si.r[i]	]BS$
0017		{vti.i=(isP[i&i])si.r[i].	BS$

Рисунок 2 Протокол работы автомата с магазинной памятью начало

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Е

```
1008   X->o[I].X                    o[i].}}                      X}}$
1008   SAVESTATE: 0087
1008                                o[i].}}                      o[I].X}}$
1009                                [i].}}                       [I].X}}$
1010                                i].}}                        I].X}}$
1011   I->i                         i].}}                        I].X}}$
1011   SAVESTATE: 0088
1011                                i].}}                        i].X}}$
1012                                ].}}                         ].X}}$
1013                                .}}                          .X}}$
1014                                }}                           X}}$
1015   TNS_NORULECHAIN/NS_NORULE
1015   RESSTATE
1015                                i].}}                        I].X}}$
1016   TNS_NORULECHAIN/NS_NORULE
1016   RESSTATE
1016                                o[i].}}                      X}}$
1017   X->oK.                       o[i].}}                      X}}$
1017   SAVESTATE: 0087
1017                                o[i].}}                      oK.}}$
1018                                [i].}}                       K.}}$
1019   K->[W]                       [i].}}                       K.}}$
1019   SAVESTATE: 0088
1019                                [i].}}                       [W].}}$
1020                                i].}}                        W].}}$
1021   W->i                         i].}}                        W].}}$
1021   SAVESTATE: 0089
1021                                i].}}                        i].}}$
1022                                ].}}                         ].}}$
1023                                .}}                          .}}$
1024                                }}                           }}$
1025                                }                            }$
1026                                                             $
1027   LENTA_END
1028   ------>LENTA_END
```

Рисунок 3 Протокол работы автомата с магазинной памятью конец

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ Е

ДЕРЕВО РАЗБОРА

```
0000   S->ftiFBS
0003   F->[P]
0004   P->ti&P
0007   P->ti
0010   B->{Nr[I].}
0011   N->vti=E.
0015   E->iM
0016   M->sE
0017   E->i
0021   I->i
0025   S->h{N}
0027   N->vti=E.N
0031   E->l
0033   N->vti=E.N
0037   E->l
0039   N->vti=E.N
0043   E->iK
0044   K->[W]
0045   W->i&W
0047   W->i
0050   N->oK.
0051   K->[W]
0052   W->i
```

Рисунок 4 Дерево разбора

ПРИЛОЖЕНИЕ Ж

```
using System;

namespace CourseProject
{
    class MMV2020
    {

        static string concat(string concat_a, string concat_b)
        {
        string concat_result = concat_a+concat_b;
        return concat_result ;
        }
        static void Main(string[] args)
        {
        string head_stra = "just ";
        string head_strb = "string";
        string head_strc = concat(head_stra, head_strb);
        Console.WriteLine(head_strc);
        }

    }
}
```

Рисунок 5 Результат генерации кода

Литература

1. Ахо, А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – М.: Вильямс, 2003. – 768с.

2. Смелов, В.В. Курс лекций по предмету языки программирования – 2016

3. Прата, С. Язык программирования C++. Лекции и упражнения / С. Прата. – М., 2006 — 1104 с.

4. Страуструп, Б. Принципы и практика использования C++ / Б. Страуструп – 2009 – 1238 с.