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 TMA-2017»

Выполнил студент Тихонович Максим Александрович

(Ф.И.О.)

Руководитель проекта ст.пр. Наркевич Аделина Сергеевна

(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 ст.пр. Наркевич Аделина Сергеевна

(учен. степень, звание, должность, подпись, Ф.И.О.)

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер ст.пр. Наркевич Аделина Сергеевна

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2017

# Содержание

[Содержание 2](#_Toc501709568)

[Введение 5](#_Toc501709569)

[Глава 1. Спецификация языка программирования 6](#_Toc501709570)

[1.1. Характеристика языка программирования 6](#_Toc501709571)

[1.2. Алфавит языка 6](#_Toc501709572)

[1.3 Применяемые сепараторы 6](#_Toc501709573)

[1.4. Применяемые кодировки 6](#_Toc501709574)

[1.5. Типы данных 7](#_Toc501709575)

[1.6. Преобразование типов данных: 7](#_Toc501709576)

[1.7. Идентификаторы 7](#_Toc501709577)

[1.8. Литералы 7](#_Toc501709578)

[1.9. Объявление данных и область видимости 7](#_Toc501709579)

[1.10. Инициализация данных 8](#_Toc501709580)

[1.11. Инструкции языка 8](#_Toc501709581)

[1.12. Операции языка 8](#_Toc501709582)

[1.13. Выражения и их вычисления 9](#_Toc501709583)

[1.14. Программные конструкции языка 9](#_Toc501709584)

[1.15. Область видимости идентификаторов 9](#_Toc501709585)

[1.16. Семантические проверки 9](#_Toc501709586)

[1.19. Ввод и вывод данных 10](#_Toc501709587)

[1.20. Точка входа 10](#_Toc501709588)

[1.21. Препроцессор 11](#_Toc501709589)

[1.22. Соглашения о вызовах 11](#_Toc501709590)

[1.23. Объектный код 11](#_Toc501709591)

[1.24. Классификация сообщений транслятора 11](#_Toc501709592)

[1.25. Контрольный пример 11](#_Toc501709593)

[Глава 2. Структура транслятора 12](#_Toc501709594)

[2.1. Компоненты транслятора их назначение и принципы взаимодействия 12](#_Toc501709595)

[2.2. Перечень входных параметров транслятора 12](#_Toc501709596)

[2.3. Перечень протоколов, формируемых транслятором и их содержания 13](#_Toc501709597)

[Глава 3. Разработка лексического анализатора 14](#_Toc501709598)

[3.1. Структура лексического анализатора. 14](#_Toc501709599)

[3.2. Контроль входных символов 14](#_Toc501709600)

[3.3. Удаление избыточных символов 14](#_Toc501709601)

[3.4. Перечень ключевых слов, сепараторов, символов операций, соответствующих им лексем и конечных автоматов 15](#_Toc501709602)

[3.5. Основные структуры данных 16](#_Toc501709603)

[3.6. Принцип обработки ошибок и их перечень 16](#_Toc501709604)

[3.7. Алгоритм лексического анализатора 16](#_Toc501709605)

[3.8. Контрольный пример 16](#_Toc501709606)

[3.9. Алгоритм лексического анализатора 16](#_Toc501709607)

[Глава 4. Разработка синтаксического анализатора 17](#_Toc501709608)

[4.1. Структура синтаксического анализатора 17](#_Toc501709609)

[4.2. Контекстно-свободная грамматика языка, описывающая синтаксис языка 17](#_Toc501709610)

[4.3 Построение конечного магазинного автомата 18](#_Toc501709611)

[4.4. Основные структуры данных 18](#_Toc501709612)

[4.5 Описание алгоритма синтаксического разбора 19](#_Toc501709613)

[4.6 Структура и перечень сообщений синтаксического анализатора 19](#_Toc501709614)

[4.7 Параметры синтаксического анализатора и режимы его работы 19](#_Toc501709615)

[4.8. Принцип обработки ошибок 19](#_Toc501709616)

[4.9. Контрольный пример 19](#_Toc501709617)

[Глава 5 Разработка семантического анализатора 20](#_Toc501709618)

[5.1. Структура семантического анализатора 20](#_Toc501709619)

[5.2. Функции семантического анализатора 20](#_Toc501709620)

[**5.3 Структура и перечень сообщений семантического анализатора** 20](#_Toc501709621)

[5.4. Принцип обработки ошибок 20](#_Toc501709622)

[5.5. Контрольный пример 21](#_Toc501709623)

[Глава 6 Преобразование выражений 22](#_Toc501709624)

[6.1 Выражения, допускаемые языком 22](#_Toc501709625)

[6.2 Польская запись и принцип её построения 22](#_Toc501709626)

[6.3. Примеры преобразования выражений 23](#_Toc501709627)

[Глава 7. Генерация кода 24](#_Toc501709628)

[Глава 8 Тестирование транслятора 26](#_Toc501709629)

[Приложение A: Контрольный пример 27](#_Toc501709630)

[Приложение Б: Таблица входных символов 28](#_Toc501709631)

[Приложение В: Графы переходов 29](#_Toc501709632)

[Приложение Г: Структуры таблицы лексем и таблицы идентификаторов 30](#_Toc501709633)

[Приложение Д: Тестирование 33](#_Toc501709634)

[Приложение Е: Грамматика 36](#_Toc501709635)

[Приложение Ж: Таблица ошибок 37](#_Toc501709636)

[Приложение З: Магазинный автомат 38](#_Toc501709637)

[Приложение И: Результат работы синтаксического анализатора 39](#_Toc501709638)

[Заключение 42](#_Toc501709639)

[Список используемой литературы 43](#_Toc501709640)

# Введение

Перед нами поставлена задача разработки языка программирования. Разрабатываемое приложение состоит из нескольких подпограмм. Каждая подпрограмма выполняется последовательно. Название разрабатываемого языка программирования TMA-2017. Для данного языка была разработан свой синтаксис, а также семантика. Запуск приложения должен осуществляться посредством командной строки с указанием параметров запуска. Основной целью курсового проекта, является усвоение устройства языков программирования и принцип работы компиляторов. Написание транслятора будет осуществляться на языке C++. Для выполнения данной задачи были поставлены следующие задачи:

1. Разработка спецификации языка программирования;

2. Разработка структуры транслятора;

3. Разработка лексического анализатора;

4. Разработка синтаксического анализатора;

5. Разработка семантического анализатора;

6. Разработка транслятора кода;

7. Преобразование выражений;

8. Тестирование транслятора;

Решения каждой из поставленных задач буду приведены в соответствующих главах курсового проекта, а именно:

1. спецификация языка программирования;
2. структура транслятора;
3. лексический анализатор;
4. синтаксический анализатор;
5. семантический анализатор;
6. преобразование выражений;
7. генерация кода;
8. тестирование транслятора.

# Глава 1. Спецификация языка программирования

## 1.1. Характеристика языка программирования

Язык программирования TMA-2017 является универсальным, строго типизированным, не объектно-ориентированным, интерпретируемым.

## 1.2. Алфавит языка

В данном языке программирования при написании кода разрешены только латинские символы размером один байт кодировки ASCII, приведенной на рисунке 1.1 ниже.

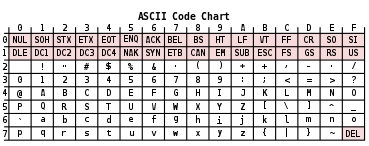


Рис. 1.1 Базовые символы языка TMA-2017

Символы, используемые на этапе выполнения: [a…z], [A…Z], [0…9]. А также спецсимволы: { } ( ) ; \* + - / = , ’ [ ] пробел. Данные символы находятся в первой половине Windows-1251.

## 1.3 Применяемые сепараторы

Язык TMA-2017 разрешает использовать сепараторы, для написания исходного кода, представленные в таблице 1.2.

Табл. 1.2 Символы-сепараторы языка TMA-2017

|  |  |
| --- | --- |
| Сепаратор | Описание |
| ; – точка с запятой | Разделитель конструкций |
| {} – фигурные скобки | Программный блок |
| () – круглые скобки. | Приоритетность операций |
| [] – квадратные скобки | Программный блок |
| , -–запятая | Разделитель параметров функции |

## 1.4. Применяемые кодировки

Применяется кодировка Windows - 1251. Которая предоставлена на рис.1.

## 1.5. Типы данных

В языке TMA-2107 используется 2 типа данных:

1) int – целые числа. Инициализация по умолчанию 0. Минимальное допустимое значение int = -65536, a максимальное значение int = 65536. В случае выхода за рамки дозволенных значений, значение будет урезаться до минимального или максимального соответственно.

2) str – строка. Инициализация по умолчанию пустой строкой. Максимальная допустимая длина строки = 255 символов.

## 1.6. Преобразование типов данных:

В языке TMA-2107 преобразование не поддерживается.

## 1.7. Идентификаторы

В языке TMA-2107 идентификаторы должны быть составлены только из символов нижнего регистра английского алфавита. Типы идентификаторов: имя переменной или функции, литерал, параметр, имя стандартной функции. Идентификатор составляется из букв английского алфавита от 1 до 10 символов, без пробелов. Максимальная длина идентификатора равна 10 символам, в противном случае урезается до указанной длины. Идентификатор не может совпадать с ключевыми словами. Максимальное количество идентификаторов 100.

## 1.8. Литералы

В языке TMA-2107 существует 2 типа литералов: литералы целого типа и строковые. Представлены в таблице. 1.2.

Табл. 1.2 Литералы

|  |  |
| --- | --- |
| Тип | Описание |
| Литерал целого типа | Числовое значение. Может иметь знак (положительное или отрицательное). Может быть только rvalue. |
| Строковый литерал | Символы, заключённые в ' ' (одинарные кавычки). Могут быть только rvalue. Могут состоять только из символов верхнего и нижнего регистра английского алфавита. |

## 1.9. Объявление данных и область видимости

В языке TMA-2107 объявление данных начинается с ключевого слова declare, указывается тип данных и имя идентификатора.

Пример: declare int a, declare str b;

Область видимости: сверху вниз, параметры внутри функции, объявления внутри функции видны только внутри функции, объявления переменных за пределами функций и главной функции не предусмотрены.

## 1.10. Инициализация данных

Инициализация переменной происходит после её объявления. Инициализация переменной в момент объявления запрещена.

Например: declare str word; word = ‘слово’; declare int num; num = 5;

## 1.11. Инструкции языка

В языке TMA-2017 предусмотрены следующие инструкции. Представлены в таблице 1.3.

Табл. 1.3 Инструкции языка TMA-2017

|  |  |
| --- | --- |
| Инициализация переменной | Имя переменной = значение; |
| Создание внешней функции | function идентификатор (тип данных идентификатор, …)  Область видимости сверху вниз (по принципу С++). Все переменные являются локальными. Предусмотрены только функции типа дынных number. |
| Главная функция | go ()  [  …  end;  ] |
| Возврат из подпрограммы | return идентификатор или литерал; |
| Вывод данных | print(идентификатор); |

## 1.12. Операции языка

В языке TMA-2017 предусмотрены следующие операции с данными. Приоритетность операций определяется с помощью (). Операции представлены в таблице. 1.4.

Табл. 1.4 Операции языка TMA-2017

|  |  |
| --- | --- |
| **Операция** | **Описание** |
| + | Бинарный, суммирование |
| - | Бинарный, разность |
| \* | Бинарный, умножение |
| / | Бинарный, деление |

## 1.13. Выражения и их вычисления

В языке TMA-2017 выражения записываются в строчку. Круглые скобки используются для изменения приоритета.

## 1.14. Программные конструкции языка

В языке программирования TMA-2017 предусмотрена одна главная функция и внешние функции. Программные конструкции представлены в таблице 1.5.

Табл. 1.5 Программные конструкции TMA-2017

|  |  |
| --- | --- |
| Внешняя функция | function идентификатор (тип данных идентификатор, … )  {  …  return идентификатор / литерал;  }  Область видимости сверху вниз. Все переменные являются локальными. |
| Главная функция | go  [  …  end;  ]  Область видимости сверху вниз. Все переменные являются локальными. |

## 1.15. Область видимости идентификаторов

Область видимости идентификаторов в языке TMA-2017 – локальная внутри программных блоков функций.

Сверху вниз, параметры внутри функции, объявления внутри функции видны только внутри функции, объявления за пределами функций и главной функции запрещены.

## 1.16. Семантические проверки

В языке программирования TMA-2017 выполняются семантические проверки. Список ошибок представлен в Приложении Д.

Семантически проверки языка TMA-2017:

1. Наличие главной функции go;
2. Проверка на попытку переопределения;
3. Проверка на несоответствие типов;
4. Проверка на наличие второй главной функции go;
5. Проверка параметров функций стандартной библиотеки;
6. Использование стандартных функций без подключения библиотеки;
7. Проверка на соответствие параметров в функции;
8. Проверка литералов.

**1.17. Распределение оперативной памяти на этапе выполнения**

Все переменные помещаются в стек. На этапе выполнения программы становятся глобальными.

**1.18. Стандартная библиотека и ее состав**

В TMA-2017 присутствует стандартная математическая библиотека math. Подключение стандартной библиотеки обязательно перед использованием функций. Пример: math; Возможные функции стандартной библиотеки описаны в таблице 1.6.

Табл. 1.6 Стандартная библиотека

|  |  |
| --- | --- |
| Функция | Описание |
| xpow(идентификатор или литерал, идентификатор или литерал); | Возводит идентификатор в степень. Применима только для идентификаторов типа переменная, тип данных number, числовых литералов. |
| fact(идентификатор или литерал); | Вычисляет факториал идентификатора. Применима только для идентификаторов типа переменная, тип данных number, числовых литералов |
| koren (идентификатор или литерал); | Получение квадратного корня. Применима только для идентификаторов типа переменная, тип данных int, числовых литералов. |

## 1.19. Ввод и вывод данных

В языке TMA-2017 вывод происходит с помощью оператора print.

Например: print(‘world’);

## 1.20. Точка входа

В языке TMA-2017 может быть только одна точка входа и определяется наличием функции **go.** При инициализации более одной или менее одной – выдаст ошибку лексического анализатора.

## 1.21. Препроцессор

Директивы препроцессора не предусмотрены.

## 1.22. Соглашения о вызовах

В языке TMA-2017 применяется собственное соглашение о вызове.

Используется по умолчанию в программе. Порядок обработки аргументов – справа - налево, через стек.

## 1.23. Объектный код

Язык программирования TMA-2017 транслируется в JavaScript.

## 1.24. Классификация сообщений транслятора

В языке TMA-2017 существует префиксная таблица классификации сообщений транслятора, описанная в таблице 1.7 и таблица с критическими ошибками, описанная в таблице 1.8.

Табл. 1.7 Префиксная классификация сообщений транслятора TMA-2017

|  |  |
| --- | --- |
| **Префикс** | **Описание** |
| [LA] | Префикс для ошибок лексического анализатора |
| [IN] | Префикс для фатальных ошибок |
| [SA] | Префикс для ошибок синтаксического анализатора |
| [SMA] | Префикс для ошибок семантического анализатора |

Табл. 1.8 Фатальные ошибки транслятора TMA-2017

|  |  |
| --- | --- |
| **Номер ошибки** | **Описание** |
| Ошибка 0 | Недопустимый код ошибки |
| Ошибка 1 | Системный сбой |
| Ошибка 2 | Параметр –in должен быть задан |
| Ошибка 3 | Превышена длина входного параметра |
| Ошибка 4 | Ошибка при открытии файла с исходным кодом (-in) |
| Ошибка 5 | Ошибка при создании файла протокола (-log) |

## 1.25. Контрольный пример

Пример программы, реализованный на языке TMA-2017 представлен в приложении A.

# Глава 2. Структура транслятора

## 2.1. Компоненты транслятора их назначение и принципы взаимодействия

Схема, демонстрирующая работу транслятора представлена на рис. 2.1.

Рисунок 2.1 Схема работы транслятора TMA-2017

## 2.2. Перечень входных параметров транслятора

В трансляторе TMA-2017 предусмотрены входные параметры. Приоритет параметров повышается слева направо, если было введено более одного параметра, для одного этапа трансляции. Максимальная разрешенная длина параметра – 80 символов. Входные параметры описаны в таблице 2.1

Табл. 2.1 Входные параметры транслятора TMA-2017

|  |  |
| --- | --- |
| **Входной параметр** | **Описание** |
| -in: | Указывает транслятору в каком месте лежит исходный код. (Обязательный параметр) |
| -out: | Указывает транслятору в какой файл выводить объектный код после этапа трансляции. (При отсутствии за основу берется параметр -in) |
| -log: | Указывает транслятору в какой файл выводить протокол работы транслятора. (При отсутствии за основу берется параметр -in) |
| -DT | Вывод таблицы разделения слов. (При отсутствии параметра, таблица выводиться только в протокол) |
| -LT | Вывод таблицы лексем в консоль.  (При отсутствии параметра, таблица выводиться только в протокол) |
| Продолжение табл. 2.1 Входные параметры транслятора TMA-2017 | |
| -IT | Вывод таблицы идентификаторов в консоль. (При отсутствии параметра, таблица выводиться только в протокол) |
| -SA | Вывод дерева разбора.( При отсутствии параметра, дерево выводиться только в протокол) |
| -NT | Вывод таблицы промежуточного кода. (При отсутствии параметра, таблица выводиться только в протокол) |

## 2.3. Перечень протоколов, формируемых транслятором и их содержания

В языке TMA-2017, транслятор формирует протокол работы, описанный в таблице 2.2.

Табл. 2.2 Описание протоколов транслятора TMA-2017

|  |  |
| --- | --- |
| **Протокол** | **Описание** |
| Лексического анализатора | Формирует в протокол таблицу лексем (ТЛ) и таблицу идентификаторов (ТИ). Каждый идентификатор ТЛ ссылается на ТИ.  ТЛ формируется как входная таблица синтаксического анализатора, ТИ формируется для всего транслятора. |
| Синтаксического анализатора | Формирует в протокол пошаговую работу магазинного автомата с деревом разбора, для последующего разбора генератором кода. |
| Генератора кода | Формирует список структур промежуточного кода по таблице лексем, полученную из лексического анализатора. Из структур формируется объектный код ассемблера |

# Глава 3. Разработка лексического анализатора

## 3.1. Структура лексического анализатора.

Схема работы лексического анализатора представлена на рисунке 3.1.

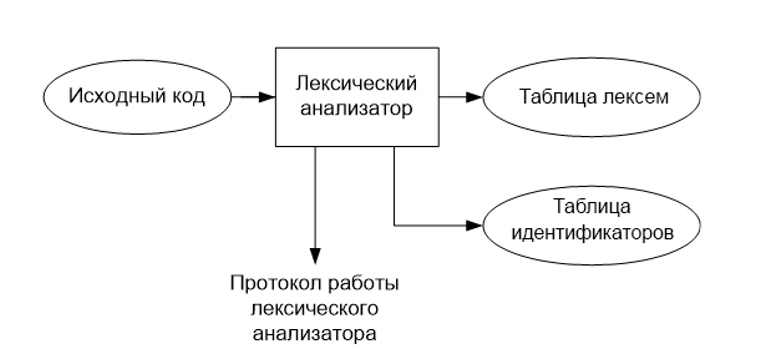


Рис. 3.1 Схема работы лексического анализатора.

Входными данными для лексического анализатора является текст программы на языке TMA-2107. В результате работы лексического анализатора получим таблицу лексем и таблицу идентификаторов.

## 3.2. Контроль входных символов

Таблица входных символов представлена на рис. 1. Реализация таблицы на языке C++ представлена в приложении Б: Таблица входных символов*.* Обозначения символов в таблице представлено в таблице 3.1.

Табл. 3.1 Контроль входных символов

|  |  |
| --- | --- |
| **Символы** | **Значение в таблице символов** |
| () \* + - = , / \ % | SEP |
| Символ новой строки | NS |
| Запрещённый | F |
| Разрешённый | T |
| Игнорируемый | I |
| Пробел | PR |

## 3.3. Удаление избыточных символов

При считывании из файла с исходным кодом в случае если считанный символ оказался пробелом, тогда мы проверяем предыдущий символ, если предыдущий символ являлся пробелом, значит текущий пробел мы просто игнорируем и не заносим в массив.

В результате работы данного алгоритма между соседними словами будет максимум 1 пробел.

## 3.4. Перечень ключевых слов, сепараторов, символов операций, соответствующих им лексем и конечных автоматов

Ключевые слова TMA-2107: declare, int, str, function, return, math, end, koren, xpow, print, go. Сепараторы TMA-2107 представлены в табл. 1. Графы переходов представлены в Приложении В: Графы переходов.

Для каждой фразы также соответствует автомат, по которому происходит разбор выражения. Автомат является – детерминированным, то есть имеет конечное состояние. Проверка происходит следующим образом: на каждый автомат в массиве подаётся фраза и с помощью графа переходов происходит разбор, если разбор выполнен, то происходит заполнение таблицы лексем и при необходимости таблицы идентификаторов.

Каждой фразе соответствует отдельный символ, называемый лексемой. Соответствие фраз с такими лексемами представлен в таблице 3.2.

Табл. 3.2 Лексемы

|  |  |  |
| --- | --- | --- |
| **Фрагмент** | **Лексема** | **Примечание** |
| int  str | v | Тип данных |
| идентификатор | i | (переменная, функция или параметр) |
| литералы | l | данные |
| function | f | функция |
| return | r | Возврат значения из функции |
| go | s | Главная функция |
| print | o | Вывод |
| koren | k | Вычисление квадратного корня |
| xpow | b | Возведение в квадрат |
| fact | a | Вычисление факториала |
| ; | ; | Сепаратор |
| { | { | Начало тела внешней функции |
| } | } | Конец тела внешней функции |
| ( | ( | Начало перечисления параметров функции |
| ) | ) | Конец перечисления параметров функции |
| = | = | Оператор присваивания |
| +  -  \*  / | +  -  \*  / | Арифметические операторы |
| , | , | Сепаратор параметров |
| Продолжение табл. 3.2 Лексемы | | |
| [ | [ | Начало тела главной функции |
| ] | ] | Конец тела главной функции |
| end | e | Завершение программы |
| declare | c | Создание переменной |

## 3.5. Основные структуры данных

Код С++ с описанием структур таблиц лексем и идентификаторов, приведён в приложении Г: Структуры таблицы лексем и таблицы идентификаторов.

## 3.6. Принцип обработки ошибок и их перечень

При обнаружении ошибки во время работы транслятора, вызывается функция получения ошибки, в которую передается, в зависимости от места возникновения ошибки, следующая информация: код ошибки, номер строки в коде, номер позиции в строке или только код ошибки.

## 3.7. Алгоритм лексического анализатора

После разбиения текста из файла с исходным кодом на слова, для каждого слова подбирается автомат способный его разобрать, в случае если такой автомат существует, тогда цепочка будет разобрана, иначе ошибка. Далее лексический анализатор анализирует лексему, соответствующую данному слову и выполняет действия описанные для данной лексемы. Лексический анализатор продолжает работать пока не будет разобрано последнее слово.

## 3.8. Контрольный пример

В результате работы лексического анализатора мы получим таблицу лексем, таблицу идентификаторов и информацию об ошибках. Текст таблицы лексем и таблицы идентификаторов представлен в Приложении Г: Структуры таблицы лексем и таблицы идентификаторов.

## 3.9. Алгоритм лексического анализатора

После разбиения текста из файла с исходным кодом на слова, для каждого слова подбирается автомат способный его разобрать, в случае если такой автомат существует, тогда цепочка будет разобрана, иначе ошибка. Далее лексический анализатор анализирует лексему, соответствующую данному слову и выполняет действия описанные для данной лексемы. Лексический анализатор продолжает работать пока не будет разобрано последнее слово.

# Глава 4. Разработка синтаксического анализатора

## 4.1. Структура синтаксического анализатора

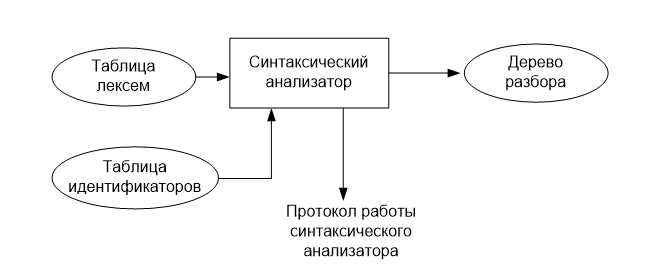
Структура работы синтаксического анализатора и входные-выходные данные представлены на рисунке. 4.1.

Рисунок 4.1 Синтаксический анализатор

## 4.2. Контекстно-свободная грамматика языка, описывающая синтаксис языка

Грамматика для синтаксического разбора языка TMA-2107 представлена

в таблице 4.1. Для описания языка, разбираемого синтаксического анализатором, применяют грамматики типа 2 – контекстно-свободные грамматики.

Грамматика типа 2:  - контекстно-свободная грамматика, где T – множество терминалов, N – множество нетерминалов, P – правила перехода, S – стартовый символ. Описание терминалов представлено в таблице 4.1.

Правила имеют вид: , где , ,  - словарь грамматики .

Табл. 4.1 Правила перехода нетерминальных символов

|  |  |
| --- | --- |
| Нетерминал | Правила |
| S – структура программы | fi(F){N}S | s[N] | m;S |
| N – конструкции внутри функции | cni;N | i=E;N | oE;N | i=E;N | m;N| i=E; | rE; | e; |
| F – параметры функции | ni | ni,F |
| E - выражения | i | l | | (E) | iM |i(W) | k(i) | k(l) | b(i) | b(l) | iM | (E)M |
| W – параметры функции при вызове | i,W | l,W | l | i |
| M - знаки | +E | +EM |-E |-EM | \*E | \*EM | /E | /EM |

## 4.3 Построение конечного магазинного автомата

Конечный автомат с магазинной памятью можно представить в виде семёрки , где М – автомат, Q – множество состояний, V – алфавит входных символов, Z – алфавит магазина,  - функция переходов,  – начальное состояние автомата,  – начальное состояние магазина, F – множество конечных состояний. Автомат с магазинной памятью представлен на рисунке 4.1



Рис. 5.1 Автомат с магазинной памятью.

Принцип работы автомата следующий:

1. в магазин записывается стартовый символ;
2. на основе полученных раннее 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 с 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 вызывается ошибка.

Структура магазинного автомата представлена в виде структуры с++ (описание автомата находится в зоне комментариев) в приложении З: Магазинный автомат.

## 4.4. Основные структуры данных

Код C++ с описанием структур данных описывающих грамматику представлен в Приложении Е: Грамматика*.*

## 4.5 Описание алгоритма синтаксического разбора

Алгоритм синтаксического разбора основан на работе автомата с магазинной памятью. Работа данного автомата описана в пункте 4.3.

## 4.6 Структура и перечень сообщений синтаксического анализатора

Перечень сообщений синтаксического анализатора представлен в таблице 4.2.

Таблица 4.2 - Перечень сообщений синтаксического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 15 | Неверная структура программы |
| 16 | Ошибка в параметрах функции |
| 17 | Ошибочный оператор |
| 18 | Ошибка в выражении |
| 19 | Ошибка в параметрах вызываемой функции |

## 4.7 Параметры синтаксического анализатора и режимы его работы

Транслятор TMA-2017 допускает использование параметров для управления работой синтаксического анализатора. Принцип их использования описан в таблице 2.1.

## 4.8. Принцип об работки ошибок

При возникновении ошибок синтаксический анализатор пытается откатиться назад, если это возможно и повторить шаг по следующей цепочке правил, в случае невозможности вызывается специализированная ошибка для правила, до которого максимально добрался анализатор. Ошибки выдаваемы синтаксическим анализатором описаны в приложении Ж: Таблица ошибок.

## 4.9. Контрольный пример

Протокол разбора и дерево разбора представлены в приложении И: Результат работы синтаксического анализатора.

# Глава 5 Разработка семантического анализатора

## 5.1. Структура семантического анализатора

Проверка на ошибки в исходном коде производится как одновременно с работой синтаксического и лексического анализаторов, так и в отдельной части транслятора. Семантические правилаязыка TMA-2107 представлены в гл. 1. п. 1.16.

## 5.2. Функции семантического анализатора

Семантический анализатор выполняет проверку на основные правила языка (семантики языка), которые представлены в гл. 1. п. 1.16*.* Таблица ошибок представлена в виде фрагмента кода C++ в приложении Ж: Таблица ошибок.

**5.3 Структура и перечень сообщений семантического анализатора**

Принцип обработки ошибок семантическим анализатором заключается в следующем: все обнаруженные ошибки записываются в общую структуру и продолжается дальнейшая работа транслятора. Сообщения, формируемые семантическим анализатором представлены в таблице 5.1.

Таблица 5.1 - Перечень сообщений семантического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 6 | Отсутствие главной функции |
| 10 | Обнаружена вторая главная функция |
| 11 | Несоответствие типов |
| 12 | Запрещено присваивать значение функции |
| 13 | Str идентификатор – параметр функции библиотеки math |
| 21 | Использование встроенной функции без подключения библиотеки |
| 22 | Попытка использовать операцию присваивания между двумя идентификаторами |
| 25 | Несоответствие параметров функции |
| 26 | Обнаружены str идентификаторы в математическом выражении |
| 28 | Неверное количество параметров вызываемой функции |

## 5.4. Принцип обработки ошибок

В случае возникновения ошибок, вызываем функцию получения ошибки, которая принимает обязательным параметром код ошибки в таблице сообщений. Затем производится вывод ошибки в поток лог-файла, в случае отсутствия данного потока, ошибка выводится в консоль. Структура данных с++ в которой хранится информация об ошибках представлена в приложении Ж: Таблица ошибок*.*

## 5.5. Контрольный пример

Контрольные примеры, предназначенные для проверки обработки всех возможных ошибок, представлены в виде текстовых файлов с исходным кодом на языке TMA-2107, которые прилагаются на переносном носителе CD-R вместе с файлами решения под среду разработки MS Visual Studio 2015. Также несколько примеров которые демонстрируют работу обработчика ошибок представлены в приложении Д: Тестирование.

# Глава 6 Преобразование выражений

## 6.1 Выражения, допускаемые языком

В языке TMA-2107 допускаются вычисления выражений исключительно с целочисленными типами данных. Вычисление выражений в языке TMA-2107 происходит без преобразования в польскую запись. Приоритет операций представлен в таблице 6.1.

Табл. 6.1 Приоритеты операций

|  |  |
| --- | --- |
| Приоритет | Операция |
| 1 | ( |
| 1 | ) |
| 2 | + |
| 2 | - |
| 3 | \* |
| 3 | / |

## 6.2 Польская запись и принцип её построения

Польская запись - это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок. Существует два типа польской записи: прямая и обратная, также известные как префиксная и постфиксная. Отличие их от классического, инфиксного способа заключается в том, что знаки операций пишутся не между, а, соответственно, до или после аргументов.

Принцип построения польской записи:

0.Рассматриваем поочередно каждый символ:

1. Если этот символ - число (или переменная), то просто помещаем его в выходную строку.

2. Если символ - знак операции (+, -, \*, / ), то проверяем приоритет данной операции.

Операции умножения и деления имеют наивысший приоритет (допустим он равен 3).

Операции сложения и вычитания имеют меньший приоритет (равен 2). Наименьший приоритет (равен 1) имеет открывающая скобка.

Получив один из этих символов, мы должны проверить стек:

а) Если стек все еще пуст, или находящиеся в нем символы (а находится в нем могут только знаки операций и открывающая скобка)

имеют меньший приоритет, чем приоритет текущего символа, то помещаем текущий символ в стек.

б) Если символ, находящийся на вершине стека имеет приоритет, больший или равный приоритету текущего символа,

то извлекаем символы из стека в выходную строку до тех пор, пока выполняется это условие; затем переходим к пункту а).

3. Если текущий символ - открывающая скобка, то помещаем ее в стек.

4. Если текущий символ - закрывающая скобка, то извлекаем символы из стека в выходную строку до тех пор,

пока не встретим в стеке открывающую скобку (т.е. символ с приоритетом, равным 1), которую следует просто уничтожить.

Закрывающая скобка также уничтожается. Если вся входная строка разобрана, а в стеке еще остаются знаки операций,

извлекаем их из стека в выходную строку.

## 6.3. Примеры преобразования выражений

Примеры преобразования выражений к польской записи представлены в таблице. 6.2.

Табл. 6.2. Примеры выражений в польской записи

|  |  |
| --- | --- |
| Выражение | Выражение в польской записи |
| 2\*(3+4) | 234+\* |
| 2+3 | 23+ |

# Глава 7. Генерация кода

Транслятор TMA-2107 выполняет трансляцию кода на языке TMA-2107 в код на языке JavaScript на основе таблицы лексем и таблицы идентификаторов. Схематично трансляция кода показана на рисунке.7.1*.* Транслятор кода начинает свою работу только в том случае если код на языке TMA-2107 прошёл предыдущие компоненты транслятора без ошибок.

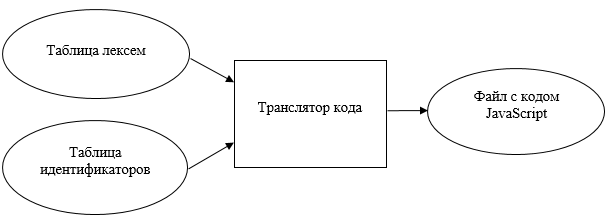


Рис. 5 Структура транслятора кода

На вход подаются по очереди лексемы, и в зависимости от поданной лексемы языка TMA-2107, в текстовый файл Generation.html записывается соответствующее ей слово. После разбора последней лексемы будет получен файл с кодом JavaScript. После открытия этого файла, откроется окно браузера и произойдёт выполнение программы. В таблице 7.1 представлены лексемы и соответствующие им слова для записи в файл.

Табл. 7.1 Трансляция JavaScript

|  |  |
| --- | --- |
| Лексема | Слово |
| { | { |
| c | var |
| ; | ; переход на новую строку |
| = | = |
| n | Запись не производится |
| пробел | Запись не производится |
| m | document.write(“Подключена математическая библиотека)” |
| o | document.write |
| ( | ( |
| ) | ) |
| } | } |
| . | Переход на новую строку |
| + | + |
| Продолжение табл. 7.1. Трансляция JavaScript | |
| - | - |
| \* | \* |
| / | / |
| f | function |
| , | , |
| r | return |
| k | koren |
| b | xpow |
| a | fact |

В случае если встречается лексема идентификатора, его имя и значение будет взято из таблицы идентификаторов.

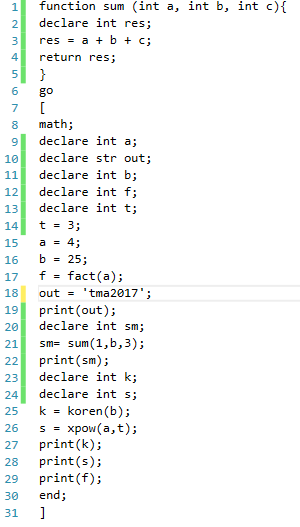
# Глава 8 Тестирование транслятора

Тестирование транслятора TMA-2107 будет выполняться посредством компиляции исходного кода в проекте Visual Studio 2015 и запуска, полученного файла с исходным кодом на языке JavaScript. Во время тестирования транслятора файл с исходным кодом TMA-2107 будет изменяться для проверки эффективности и работоспособности транслятора в различных ситуациях.

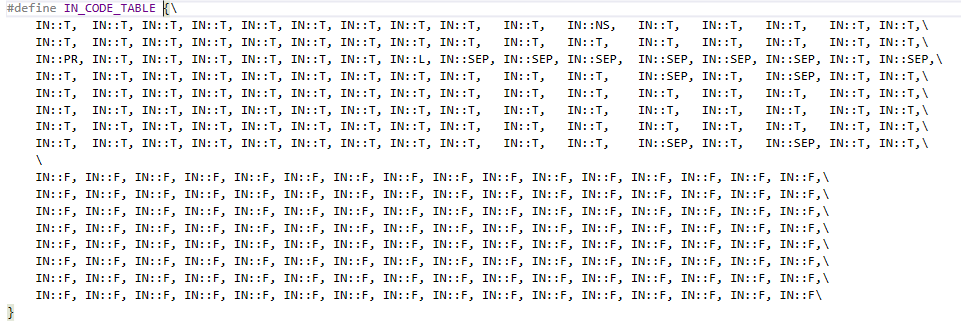
В приложении Д приведены примеры тестирования транслятора на ошибки, формируемые лексическим, синтаксическим и семантическим анализатором, а также ошибки, возникающие при открытии файла и задании параметров.

Файлы решения под среду разработки MS Visual Studio 2015, а также файлы с исходным кодом на языке TMA-2107 прилагаются на переносном носителе CD-R.

# Приложение A: Контрольный пример

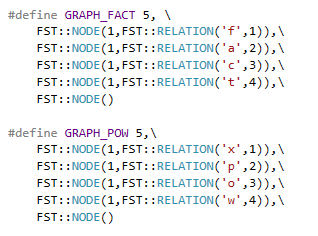


# Приложение Б: Таблица входных символов

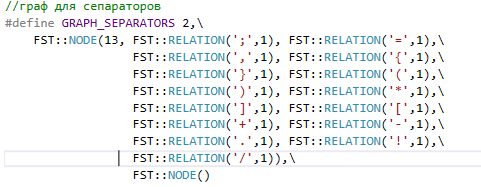


# Приложение В: Графы переходов

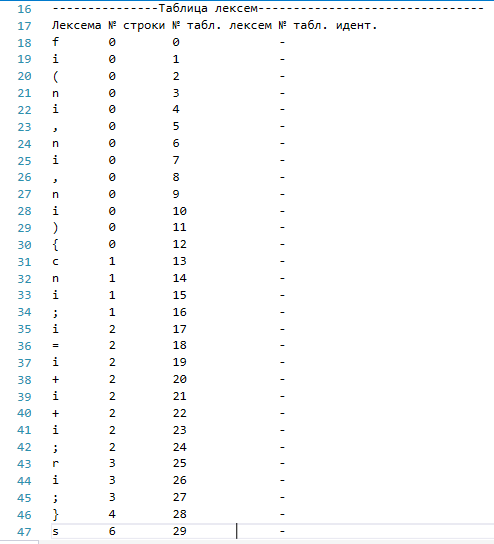
Графы функций стандартной библиотеки:

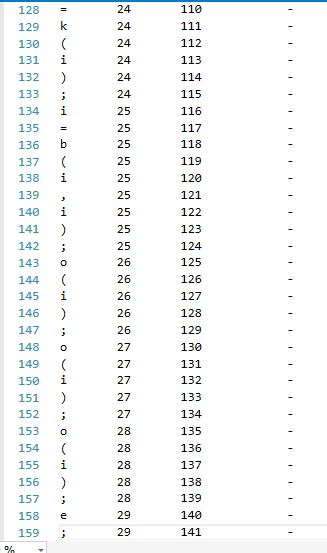


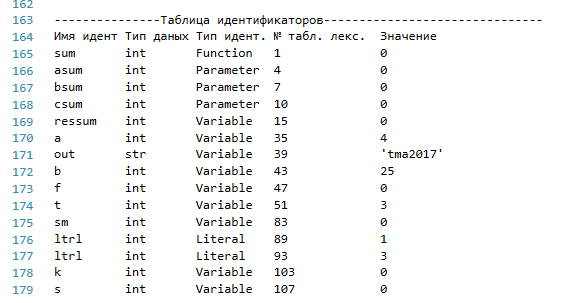
Граф для сепараторов:

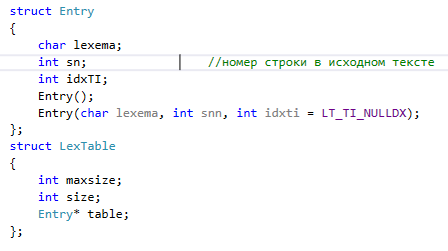


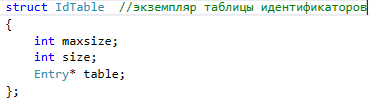
# Приложение Г: Структуры таблицы лексем и таблицы идентификаторов

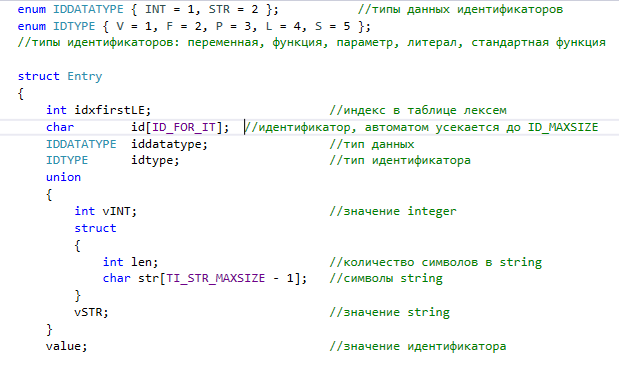




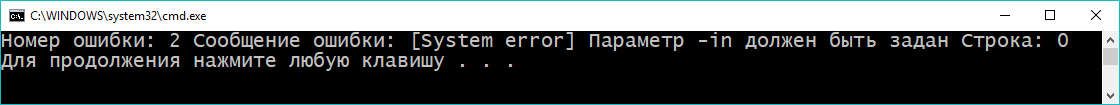


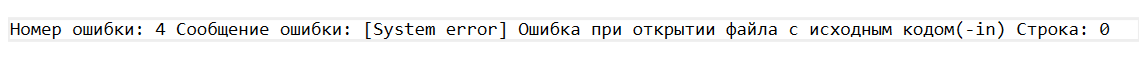




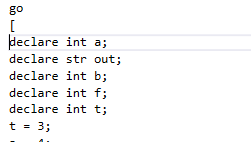


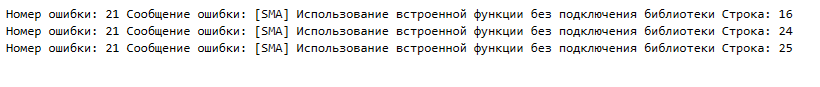
# Приложение Д: Тестирование

При отсутствии входного файла возникает ошибка с номером 2:

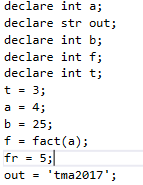
Если путь для входного файла задан неправильно, то возникает ошибка с номером 4:

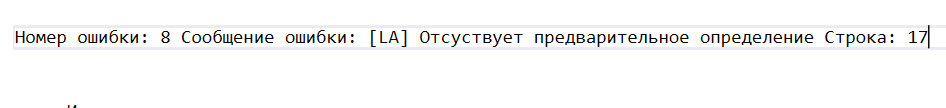
При использовании функций из стандартной библиотеки без ее подключения возникает ошибка под номером 21:



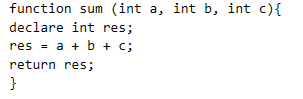


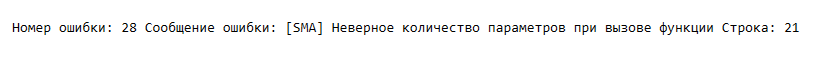
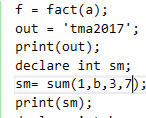
При присваивании переменной значения без предварительного определения возникает ошибка под номером 8:





При вызове функции с неверным количеством параметров возникает ошибка с номером 28:





При присвоении значению одного типа значению другого типа возникает ошибка с номером 11:

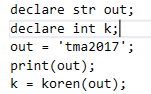




При лишних параметрах математической функции возникает ошибка с номером 18:



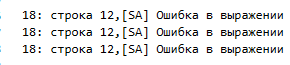
При передачи значения строкового типа в математическую функцию возникает ошибка с номером 13:



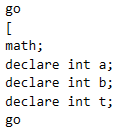


При использовании неверного оператора возникает ошибка с номером 18:



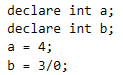


При наличии двух главных функций go возникает ошибка с номером 10:





При делении на ноль возникает ошибка с номером 29:





При добавлении лишней скобки возникает ошибка с номером 18:



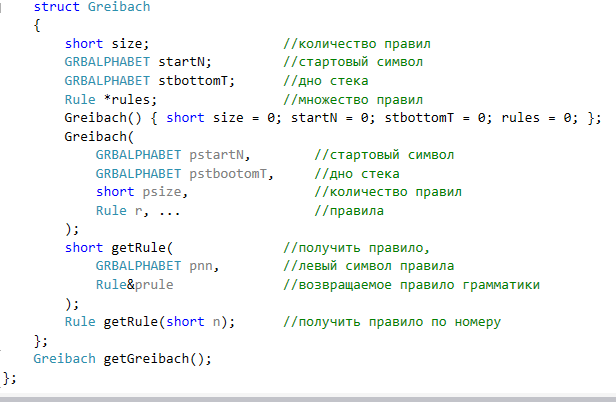


При превышении длины строкового литерала возникает ошибка с номером 24:

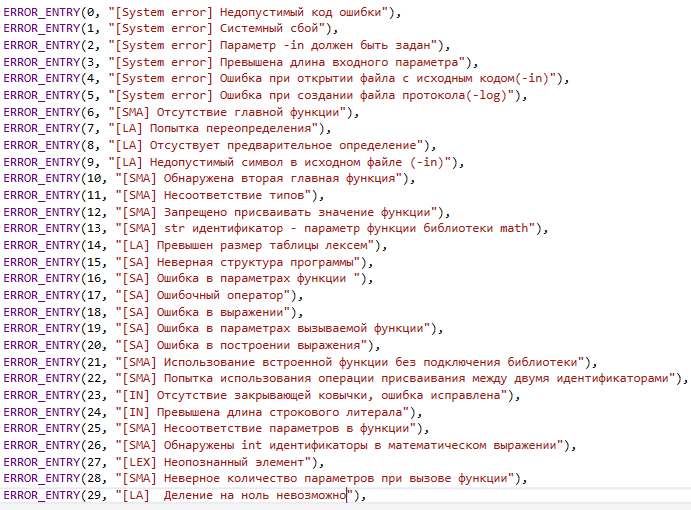




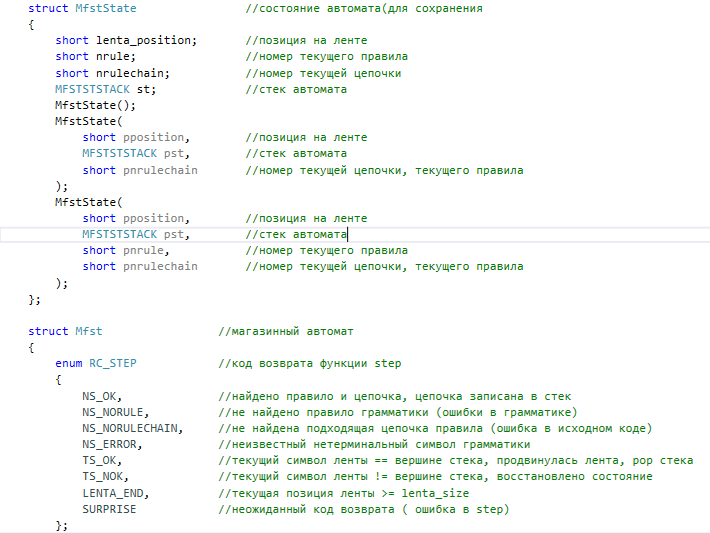
# Приложение Е: Грамматика



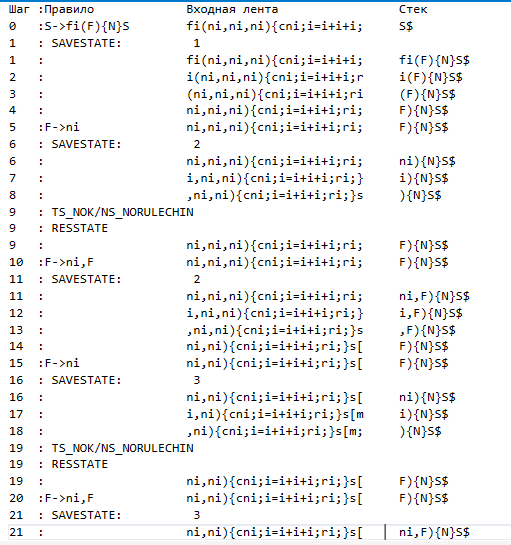
# Приложение Ж: Таблица ошибок

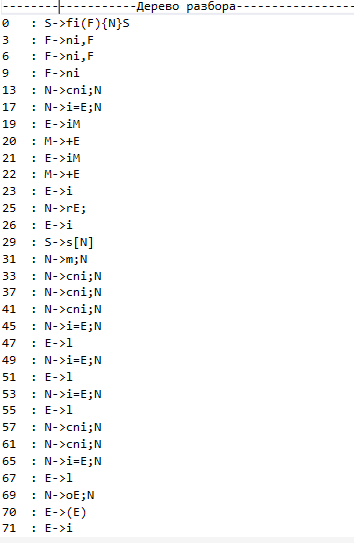


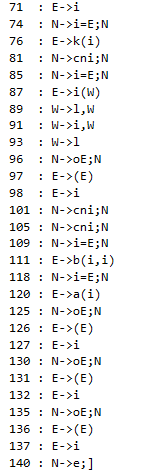
# Приложение З: Магазинный автомат



# Приложение И: Результат работы синтаксического анализатора







# Заключение

В данном курсовом проекте были выполнены поставленные минимальные требования. В ходе работы было изучено много нового, а также закреплены знания, которые были получены ранее. Также стоит отметить что данный курсовой проект позволил совместить закрепление знаний сразу по двум языкам программирования, таких как C++ и JavaScript. При написании приложения были усвоены такие понятия как синтаксический, лексический и семантический анализатор.

В итоге был получен примитивный язык программирования TMA-2107, который не имеет сложных конструкций, которые реализованы на сегодняшний день во многих других языках программирования.

Окончательная версия языка TMA-2107 включает:

1. 2 типа данных;
2. Поддержка оператора вывода;
3. Возможность подключения и вызова функций стандартной библиотеки;
4. Наличие 4 арифметических операторов для вычисления выражений;

Возможность вызова функции в выражении.

# Список используемой литературы

1. Р. Лафоре Программирование C++ - 2004
2. Cody Lindley Введение в JavaScript – 2008
3. Ахо А, Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – М.: Вильямс 2003. – 768с.