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 KAA-2019»

Выполнил студент Конопацкий Александр Александрович

(Ф.И.О.)

Руководитель проекта преп.-стаж. Котович Дмитрий Васильевич

(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 Васильевич

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

Нормоконтролер преп.-стаж. Котович Дмитрий Васильевич

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

Курсовой проект защищен с оценкой

Минск 2019

[Введение 3](#_Toc27656904)

[Глава 1 Спецификация языка программирования 4](#_Toc27656905)

[**1.1 Характеристика языка программирования** 4](#_Toc27656906)

[**1.2 Алфавит языка** 5](#_Toc27656907)

[**1.3 Применяемые сепараторы** 6](#_Toc27656908)

[**1.4 Применяемые кодировки** 7](#_Toc27656909)

[**1.5 Типы данных** 7](#_Toc27656910)

[**1.6 Преобразование типов данных** 7](#_Toc27656911)

[**1.7 Идентификаторы** 7](#_Toc27656912)

[**1.8 Литералы** 8](#_Toc27656913)

[**1.9 Объявление данных и область видимости** 8](#_Toc27656914)

[**1.10 Инициализация данных** 8](#_Toc27656915)

[**1.11 Инструкции языка** 9](#_Toc27656916)

[**1.12 Операции языка** 10](#_Toc27656917)

[**1.13 Выражения и их вычисления** 11](#_Toc27656918)

[**1.14 Программные конструкции языка** 11](#_Toc27656919)

[**1.15 Область видимости идентификаторов** 11](#_Toc27656920)

[**1.16 Семантические проверки** 12](#_Toc27656921)

[**1.17 Распределение оперативной памяти на этапе выполнения** 12](#_Toc27656922)

[**1.18 Стандартная библиотека и ее состав** 12](#_Toc27656923)

[**1.19 Ввод и вывод данных** 13](#_Toc27656924)

[**1.20 Точка входа** 13](#_Toc27656925)

[**1.21 Препроцессор** 13](#_Toc27656926)

[**1.22 Соглашения о вызовах** 13](#_Toc27656927)

[**1.23 Объектный код** 13](#_Toc27656928)

[**1.24 Классификация сообщений транслятора** 14](#_Toc27656929)

[**1.25 Контрольный пример** 14](#_Toc27656930)

[Глава 2 Структура транслятора 15](#_Toc27656931)

[**2.1 Компоненты транслятора их назначение и принципы взаимодействия** 15](#_Toc27656932)

[**2.2 Перечень входных параметров транслятора** 16](#_Toc27656933)

[**2.3 Перечень протоколов, формируемых транслятором и их содержимое** 17](#_Toc27656934)

[Глава 3 Разработка лексического анализатора 18](#_Toc27656935)

[**3.1 Структура лексического анализатора** 18](#_Toc27656936)

[**3.2 Контроль входных символов** 18](#_Toc27656937)

[**3.3 Удаление избыточных символов** 19](#_Toc27656938)

[**3.4 Перечень ключевых слов, сепараторов, символов операций соответствующим им лексемам и конечных автоматов** 19](#_Toc27656939)

[**3.5 Основные структуры данных** 20](#_Toc27656940)

[**3.6 Принцип обработки ошибок** 22](#_Toc27656941)

[**3.7 Структура и перечень сообщений лексического анализатора** 22](#_Toc27656942)

[**3.8 Параметры лексического анализатора и режимы его работы** 22](#_Toc27656943)

[**3.9 Алгоритм лексического анализа** 22](#_Toc27656944)

[**3.10 Контрольный пример** 23](#_Toc27656945)

[Глава 4 Разработка синтаксического анализатора 24](#_Toc27656946)

[**4.1 Структура синтаксического анализатора** 24](#_Toc27656947)

[**4.2 Контекстно-свободная грамматика, описывающая синтаксис языка** 24](#_Toc27656948)

[**4.3 Построение конечного магазинного автомата** 25](#_Toc27656949)

[**4.4 Основные структуры данных** 26](#_Toc27656950)

[**4.5 Описание алгоритма синтаксического разбора** 26](#_Toc27656951)

[**4.6 Структура и перечень сообщений синтаксического анализатора** 27](#_Toc27656952)

[**4.7 Параметры синтаксического анализатора и режимы его работы** 27](#_Toc27656953)

[**4.8 Принцип обработки ошибок** 27](#_Toc27656954)

[**4.9 Контрольный пример** 28](#_Toc27656955)

[Глава 5 Разработка семантического анализатора 29](#_Toc27656956)

[**5.1 Структура семантического анализатора** 29](#_Toc27656957)

[**5.2 Структура и перечень сообщений семантического анализатора** 29](#_Toc27656958)

[**5.3 Принцип обработки ошибок** 30](#_Toc27656959)

[Глава 6 Вычисление выражений 31](#_Toc27656960)

[**6.1 Выражения, допускаемые языком** 31](#_Toc27656961)

[**6.2 Польская запись и принцип ее построения** 31](#_Toc27656962)

[**6.3 Программная реализация обработки выражений** 32](#_Toc27656963)

[**6.4 Контрольный пример** 32](#_Toc27656964)

[Глава 7 Генерация кода 33](#_Toc27656965)

[**7.1 Структура генератора кода** 33](#_Toc27656966)

[**7.2 Представление типов данных в оперативной памяти** 33](#_Toc27656967)

[**7.3 Статическая библиотека** 34](#_Toc27656968)

[**7.4 Особенности алгоритма генерации кода** 34](#_Toc27656969)

[**7.5 Входные параметры генератора кода** 35](#_Toc27656970)

[**7.6 Контрольный пример** 35](#_Toc27656971)

[Глава 8 Тестирование транслятора 36](#_Toc27656972)

[**8.1 Общие положения** 36](#_Toc27656973)

[**8.2 Результаты тестирования** 36](#_Toc27656974)

[Заключение 37](#_Toc27656975)

[Приложение А 38](#_Toc27656976)

[Приложение Б 39](#_Toc27656977)

[Приложение Г 46](#_Toc27656978)

[Приложение Д 48](#_Toc27656979)

[Приложение Е 50](#_Toc27656980)

[Приложение Ж 51](#_Toc27656981)

[Литература 53](#_Toc27656982)

# **Введение**

Задачей данного курсового проекта является разработка компилятора для языка программирования KAA-2019.

Компилятор — это частный случай (разновидность) трансляторов. А транслятор — это программа, принимающая на вход программу на одном языке (в нашем случае языке KAA-2019), и преобразующая её в программу на другом языке (в нашем случае транслируя в JavaScript, в следствии чего потом будет интерпретироваться браузером).

Интерпретация — построчный анализ, обработка и выполнение исходного кода программы или запроса.

Исходя из цели курсового проекта, были определены следующие задачи:

* разработать спецификацию языка программирования;
* разработать структуру транслятора;
* разработать лексический, синтаксический и семантический анализаторы;
* преобразовать выражения;
* произвести генерацию кода, т.е. транслировать язык программирования KAA-2019 в JavaScript;
* произвести тестирование.

Решения каждой из поставленных задач будут приведены в соответствующих главах курсового проекта.

В первой главе работы определена спецификация языка программирования, т.е. описан синтаксис и семантика языка.

Во второй главе работы представлена структура транслятора, т.е. перечислены компоненты транслятора, их назначение и принципы взаимодействия, перечень входных параметров, перечень протоколов, формируемых транслятором и их содержимое.

В третьей главе работы показана разработка лексического анализатора, порождающего таблицы лексем и идентификаторов.

В четвертой главе работы рассказывается о синтаксическом анализаторе, который выполняет синтаксический разбор текста с распечаткой протокола разбора и дерева разбора на основе таблицы лексем.

В пятой главе описан семантический анализатор.

В шестой главе решены вопросы преобразования выражений, допускаемых языком и приведена часть протокола для контрольного примера, отображающая результаты преобразования выражений в польский формат.

В седьмой главе представлена генерация кода, где из промежуточного представления порождается код на целевом языке.

В восьмой главе описывается тестирование транслятора.

# **Глава 1 Спецификация языка программирования**

## **Характеристика языка программирования**

Язык программирования KAA-2019 предназначен для выполнения простейших арифметических действий и операций над строками.

Язык программирования KAA-2019 является процедурным, строго типизированным, не объектно-ориентированным, компилируемым.

Процедурный язык программирования — язык высокого уровня, в котором используется метод разбиения программ на отдельные связанные между собой модули — подпрограммы (процедуры и функции).

Строго типизированный язык программирования — язык, в котором переменные привязаны к конкретным типам данных. Язык не позволяет смешивать в выражениях различные типы и не выполняет автоматические неявные преобразования.

Объектно-ориентированный язык программирования — язык, построенный на принципах объектно-ориентированного программирования. В основе концепции объектно-ориентированного программирования лежит понятие объекта — некой сущности, которая объединяет в себе поля (данные) и методы (выполняемые объектом действия).

Компилируемый язык программирования — язык программирования, исходный код которого преобразуется компилятором в исходный код на другом языке программирования.

## **Алфавит языка**

Алфавит языка KAA-2019 основан на кодировке Windows-1251. Таблица кодировки Windows-1251 представлена на рисунке 1.1.

Исходный код может содержать символы латинского алфавита, цифры десятичной системы счисления от 0 до 9, русские символы разрешены только в строковых литералах.



Рисунок 1.1 — Основная таблица Windows-1251

## **Применяемые сепараторы**

Символы-сепараторы – символы, используемые для разделения отдельных лексических единиц или функциональных элементов в исходном коде программы. Символы, которые являются сепараторами представлены в таблице 1.1.

Таблица 1.1 - Применяемые сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение сепаратора |
| .  “ “ (пробел)  , | Символы-сепараторы для разделения инструкций, параметров функции |
| [  ] | Символы-сепараторы указывающие на программный блок |

Продолжение таблицы 1.1

|  |  |
| --- | --- |
| (  ) | Символы-сепараторы указывающие на параметры/приоритетность операций(в выражениях) |

## **Применяемые кодировки**

Для написания исходного кода на языке программирования KAA-2019 используется кодировка Windows-1251, которая представлена на рисунке 1.1.

## **Типы данных**

В язык KAA-2019 предусмотрены три типа данных: беззнаковый целый (1 байт), строковый и логический, представленные в таблице 1.2.

Таблица 1.2 — Типы данных

| Тип | Описание |
| --- | --- |
| byte | Целочисленный тип данных. В памяти занимает 1 байт. Предназначен для арифметических операций над числами.  Автоматическая инициализация 0 (нулем). Максимальное значение 255, минимальное 0. |
| string | Строковый тип данных. Один символ занимает 1 байт. Максимальная длина строки (число символов) составляет 254 символа. При объявлении автоматически инициализируется длинной 0, и концом строки “\0”. |
| bool | Логический тип данных. Предусмотрен для объявления логических данных (1 байт). Автоматическая инициализация не используется. |

## **Преобразование типов данных**

Преобразование типов данных в языке KAA-2019 не поддерживается.

## **Идентификаторы**

Идентификатор — имя компонента программы (переменной или функции), составленное программистом по определенным правилам.

В идентификаторах языка KAA-2019 разрешается использовать латинские буквы только нижнего регистров.

Идентификаторы не могут совпадать с ключевыми словами. Данные правила справедливы как для переменных, так и для функций.

Примеры правильных идентификаторов: newstr, var и т.д.

Примеры неправильных идентификаторов: if , Var, num\_1 и т.д.

## **Литералы**

Литерал — запись в исходном коде компьютерной программы, представляющая собой фиксированное значение. Литералами также называют представление значения некоторого типа данных.

В языке KAA-2019 существует 3 вида литералов: литералы целого типа, строковые и логические (true и false), описаны в таблице 1.3.

Таблица 1.3 – Литералы

|  |  |
| --- | --- |
| Тип | Описание |
| Литералы целого типа | Интерпретируются как byte, являются rvalue. Задаются в десятичной форме, либо восьмеричной. Должны быть неотрицательными. |
| Строковые литералы | Интерпретируются как string, заключаются в двойные кавычки (“hello”), являются rvalue. |
| Литералы логического типа | Интерпретируются как bool, принимают значения true и false. |

## **Объявление данных и область видимости**

Область видимости «сверху вниз» (по принципу С++). В языке KAA-2019 требуется обязательное объявление переменной перед её использованием, объявление данных начинается с указания типа данных, затем указывается имя идентификатора.

Примеры: byte a, string b, bool c;

## **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 идентификаторы, литералы, выражения и вызовы функций. Для инициализации целочисленных литералов, представленных в восьмеричной системе счисления, предусмотрен префикс 00, который ставится перед литералом.

## **Инструкции языка**

Инструкции языка KAA-2019 представлена в таблице 1.4.

Таблица 1.4 – Инструкции языка

|  |  |
| --- | --- |
| Инструкция | Форма записи |
| Объявление переменной | <тип данных> <идентификатор>. |
| Главная функция | program  [  /программный блок/  ] |
| Объявление функции | <тип данных> func <идентификатор> (<тип данных> <идентификатор>, …)  [  / программный блок /  return <идентификатор/литерал>.  ] |
| Вызов функции | <идентификатор> (<идентификатор>, ...) |
| Присвоение значения | : |
| Печать данных | print <литерал/идентификатор>. |
| Функции стандартной библиотеки  (применяются при инициализации) | strcpy(string) — копирует содержиое из строки srring.  strcat(string1, string2) — производит конкатенацию двух строк. |
| Возвращаемое значение | return <литерал/идентификатор>. |

Продолжение таблицы 1.4

|  |  |
| --- | --- |
| Инструкция условного оператора | if (условие)  [  <блок операций1>  ]  else  [  <блок операций2>  ] |

## **Операции языка**

Арифметические операции, которые можно использовать в языке KAA-2019, представлены в таблице 1.5.

Таблица 1.5 — Операции языка

|  |  |
| --- | --- |
| Инструкция | Форма записи |
| Арифметические операции языка | + — бинарное сложение  − — бинарный минус  \* — бинарное произведение  / — бинарное деление  ( ) — приоритет операций |
| Побитовые операции языка | | — побитовое или  & — побитовое и  ~ — побитовая инверсия |

Приоритет подчиняется правилам:

* если в вы­ра­же­ние без ско­бок вхо­дят толь­ко сло­же­ние и вы­чи­та­ние или толь­ко умно­же­ние и де­ле­ние, то дей­ствия вы­пол­ня­ют в том по­ряд­ке, в каком они на­пи­са­ны;
* если в вы­ра­же­ние без ско­бок вхо­дят не толь­ко дей­ствия сло­же­ния и вы­чи­та­ния, но и умно­же­ния и де­ле­ния, или оба этих дей­ствия, то сна­ча­ла вы­пол­ня­ют по по­ряд­ку (слева на­пра­во) умно­же­ние и де­ле­ние, а затем сло­же­ние и вы­чи­та­ние;
* если в вы­ра­же­нии име­ют­ся скоб­ки, то сна­ча­ла вы­чис­ля­ют зна­че­ние вы­ра­же­ний в скоб­ках.

## **Выражения и их вычисления**

Правила для выражения и их вычисления в языке KAA-2019:

* выражение записывается только в одну строку;
* операторы разделяются пробелами;
* допускается использование скобок (для изменения приоритета);
* не допускается запись двух подряд идущих арифметических операций;

## **Программные конструкции языка**

Основные программные конструкции языка KAA-2019 представлены в таблице 1.6.

Таблица 1.6 — Основные конструкции языка

|  |  |
| --- | --- |
| Конструкция | Реализация |
| Главная функция  (точка входа) | program  [  / программный блок /  ] |
| Функции | <тип данных> func <идентификатор>  (<тип данных> <идентификатор>, …)  [  / программный блок /  return <идентификатор/литерал>.  ]  <тип данных> func <идентификатор> ()  [  return <идентификатор/литерал>.  ] |

## **Область видимости идентификаторов**

Область видимости в языке KAA-2019 - «сверху вниз» (по принципу С++). Это значит, что к каждому идентификатору можно обратиться только ниже описания его описания или объявления. «Ниже» понимается буквально — ниже по тексту программы. Т.е., например, до описания функции ее нельзя вызывать.

Все переменные в языке KAA-2019 обязаны находится внутри программного блока функций. Объявление глобальных переменных (вне программных блоков функций) не предусмотрено.

## **Семантические проверки**

Основные семантические правила языка KAA-2019 проверяемые на этапах работы транслятора, представлены в таблице 1.7.

Таблица 1.7 — Семантические правила

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Должна присутствовать точка входа program и только одна |
| 3 | Не должно быть объявлений идентификаторов с одинаковыми именами в одном и том же блоке кода |
| 4 | Присваивать значение идентификатору можно только соответствующего типа |
| 5 | Вызов функции обязует использование скобок после ее названия с передачей параметров соответствующих типов или без них |
| 6 | Тип возвращаемого функцией значения должен соответствовать типу функции |
| 7 | Деление на ноль запрещено |
| 8 | Присваивать значение функции запрещено |
| 9 | Проводить арифметические операции со строковым типом данных запрещено |
| 10 | Неверно введено восьмеричное представление числа |

## **Распределение оперативной памяти на этапе выполнения**

На этапе выполнения все переменные данного языка помещаются в стек.

## **Стандартная библиотека и ее состав**

В языке KAA-2019 предусмотрена стандартная библиотека. Функции данной стандартной библиотеки применяются при инициализации. Описание функций стандартной библиотеки языка KAA-2019 представлено в таблице 1.8.

Таблица 1.8 — Библиотека

|  |  |
| --- | --- |
| Функция | Описание |
| strcpy(string); | Копирует содержимое строки string. Применима только для идентификаторов типа string, |
| strcat(string, string); | Производит конкатенацию двух строк. Применима только для идентификаторов типа string, |

## **Ввод и вывод данных**

Ввод данных языком программирования KAA-2019 не поддерживается.

Для вывода данных используется функция print (<имя идентификатора>).

Пример: print (a);

* 1. **Точка входа**

Функция точки входа в языке программирования KAA-2019 представлена в таблице 1.9.

Таблица 1.9 — Точка входа

|  |  |
| --- | --- |
| Конструкция | Реализация |
| Главная функция  (точка входа) | program  [  / программный блок /  ] |

## **Препроцессор**

Препроцессор, принимающий и выдающий некоторые данные на вход транслятору, в языке KAA-2019 отсутствует.

## **Соглашения о вызовах**

Соглашение о вызовах — это протокол для передачи аргументов функциям. Другими словами, это договоренность между вызывающим и вызываемым кодом.

В языке KAA-2019 по умолчанию применяется соглашение \_stdcall, где параметры помещаются в стек, передача параметров происходит справа налево, стек освобождает вызываемый код, возврат через регистр EAX.

## **Объектный код**

Исходный код, написанный на языке KAA-2019, транслируется нашим компилятором в язык JavaScript, после чего интерпретируется браузером.

## **Классификация сообщений транслятора**

В ходе своей работы, транслятор генерирует сообщения, которые информируют пользователя о допущенных ошибках. Все сообщения транслятора разделены на интервалы, в зависимости от того на каком этапе была обнаружена ошибка. Все интервалы представлены в таблице 1.10.

Таблица 1.10 – Сообщения транслятора

|  |  |
| --- | --- |
| Интервал | Описание |
| 100-104 | Ошибки входных параметров |
| 106-112 | Ошибки открытия и чтения файлов |
| 113-122 | Ошибки на этапе лексического анализа |
| 600-609 | Ошибки на этапе синтаксического анализа |
| 700-711 | Ошибки на этапе семантического анализа |

## **Контрольный пример**

Контрольный пример языка программирования KAA-2019 представлен в приложении А.

# **Глава 2 Структура транслятора**

## **Компоненты транслятора их назначение и принципы взаимодействия**

Транслятор языка KAA-2019 преобразует исходный код программы в код целевого языка. Процесс трансляции состоит из фаз: лексический анализ, синтаксический анализ, семантический анализ и генерация кода. Алгоритм выполнения и описание каждой фазы представлено в таблице 2*.*1. На всех фазах трансляции применяется таблица идентификаторов и таблица лексем. Графически схема транслятора представлена на рисунке 2.1.



Рисунок 2.1 — Схема структуры транслятора

Таблица 2.1 — Структура транслятора

|  |  |
| --- | --- |
| Наименование подпрограммы | Предназначение |
| Лексический анализатор | Обрабатывает входной файл исходного кода, проверяя его на разрешённые, запрещённые и игнорируемые символы. Преобразует исходный код в более простой с помощью замены длинных слов (идентификаторов, типов данных, ключевых слов и т.д.) на лексемы, состоящие из одного символа, что упрощает последующую работу с кодом. После обработки входного файла формируется таблица лексем и таблица идентификаторов. |

Продолжение таблицы 2.1

|  |  |
| --- | --- |
| Синтаксический анализатор | Проверяет правильность написанных конструкций по заданной грамматике, выявляет синтаксические ошибки при их наличии и формирует дерево разбора. |
| Семантический анализатор | Проверяет правильность исходного кода с точки зрения семантики. Реализован как отдельный этап в данном трансляторе. Проверка по правилам осуществляется после лексического и синтаксического анализа. |
| Генератор кода | Этап транслятора, выполняющий генерацию кода на языке KAA-2019 в код на языке JavaScript на основе данных, полученных на предыдущих этапах трансляции. |

## **Перечень входных параметров транслятора**

Благодаря входным параметрам транслятора осуществляется контроль за выводом протоколов работы разных компонент, а также указывается файл с исходным кодом — программы. Перечень входных параметров представлена в таблице 2.2.

Таблица 2.2 — Входные параметры транслятора

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Тип параметра |
| -in: | Указывает транслятору путь к исходному коду. | Обязательный. |
| -log: | Указывает транслятору в какой файл выводить протокол работы транслятора. | Не обязательный. |
| -lx: | Указывает транслятору в какой файл выводить результат работы лексического анализатора, а именно таблицу лексем и таблицу идентификаторов. | Не обязательный. |
| -id: | Указывает транслятору в какой файл выводить таблицу идентификаторов | Не обязательный. |
| -js: | Указывает транслятору в какой файл выводить результат работы компилятора | Не обязательный. |
| -rl: | Указывает транслятору в какой файл выводить результат работы синтаксического анализатора, а именно трассировку и дерево разбора. | Не обязательный. |

## **Перечень протоколов, формируемых транслятором и их содержимое**

Транслятор языка KAA-2019 формирует один протокол его работы. Информация, записываемая в протокол, представлена в таблице 2.3.

Таблица 2.3 — Протокол транслятора.

|  |  |
| --- | --- |
| Тип информации | Описание информации |
| Дата и время трансляции | Выводится дата и время. |
| Параметры командой строки | Выводится информация об указанных параметрах командной строки. |
| Полная таблица лексем | Выводится таблица лексем с информацией к каждой лексеме. |
| Таблица идентификаторов | Выводится таблица идентификаторов с дополнительной информацией. |
| Промежуточный код | Выводится исходный код, представленный в виде лексем. |
| Трассировочная информация синтаксического анализа | Выводится полная информация о разборе таблицы лексем на синтаксическом анализаторе. |
| Дерево разбора | Выводится правила, по которым осуществился разбор исходного кода. |
| Промежуточный код с использованием польской записи | Выводится исходный код, представленный в виде лексем, выражения представлены в польской записи. |

# **Глава 3 Разработка лексического анализатора**

## **Структура лексического анализатора**

Лексический анализатор — часть транслятора, выполняющая лексический анализ, т.е. преобразующая исходный текст, заменяя лексические единицы языка их внутренним представлением — лексемами. Входными данными для лексического анализатора является предварительно обработанный текст программы на языке KAA-2019. В результате работы лексического анализатора формируется таблица идентификаторов и таблица лексем, модифицируется протокол работы транслятора. Структура лексического анализатора KAA-2019 представлена на рисунке 3.1.

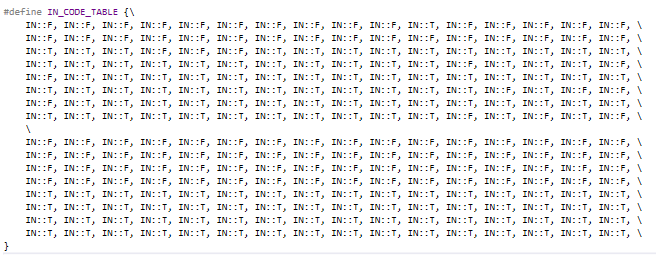


Рисунок 3.1 — Структура лексического анализатора KAA-2019

* 1. **Контроль входных символов**

Таблица для контроля входных символов представлена на рисунке 3.2.

Рисунок 3.2 — Структура лексического анализатора KAA-2019



Принцип работы таблицы: каждому элементу соответствует значение в шестнадцатеричной системе счисления - такое же, как и в таблице Windows-1251 (см. рисунок 1.1).

В представленной таблице: F — запрещённый символ, T — разрешённый символ, I — игнорируемый символ.

## **Удаление избыточных символов**

Удаление избыточных символов в языке в языке KAA-2019 не поддерживается.

## **Перечень ключевых слов, сепараторов, символов операций соответствующим им лексемам и конечных автоматов**

Перечень ключевых слов, сепараторов, символов операций соответствующих им лексем представлен в таблице 3.1.

Таблица 3.1 - Перечень ключевых слов

|  |  |
| --- | --- |
| Цепочка | Лексема |
| byte | t |
| string | t |
| bool | f |
| print | p |
| return | r |
| program | m |
| if | w |
| else | e |
| func | f |
| . | . |
| [ | [ |
| ] | ] |
| ( | ( |
| ) | ) |
| > | > |

Продолжение таблицы 3.1

|  |  |
| --- | --- |
| < | < |
| = | = |
| : | : |
| + | + |
| - | - |
| \* | \* |
| / | / |
| & | & |
| | | | |
| ~ | ~ |
| ; | ; |
| идентификатор | i |
| числовой литерал | l |
| строковый литерал | l |
| логический литерал | l |
| strcat, strcpy | s |

* 1. **Основные структуры данных**

Описание основных структур данных, используемых для хранения таблиц лексем, представлено на рис. 3.3.

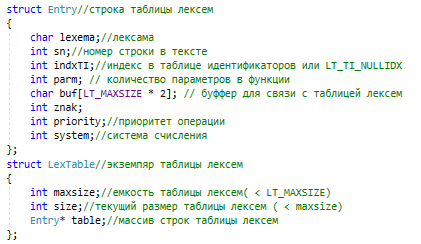


Рисунок 3.3 — Структуры таблиц лексем KAA-2019

Описание основных структур данных, используемых для хранения таблиц идентификаторов, представлено на рис. 3.4.

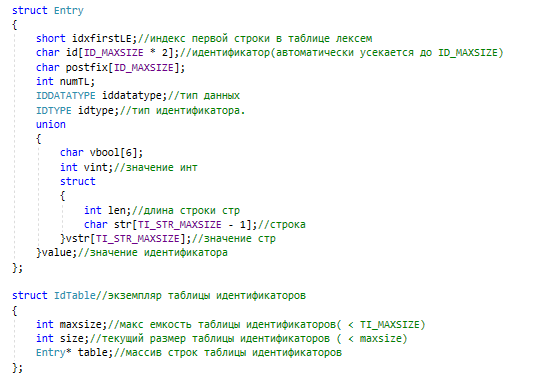


Рисунок 3.4. — Структуры таблиц идентификаторов KAA-2019

## **Принцип обработки ошибок**

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 и сообщение о типе ошибки. При возникновении ошибки работа транслятора прекращается.

## **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 |
| 113 | Превышен максимальный размер таблицы лексем |
| 114 | Превышен максимальный размер таблицы идентификаторов |
| 115 | Недопустимы номер возвращаемой строки таблицы лексем |
| 117 | Отсутствует таблица лексем |
| 118 | Отсутствует таблица идентификаторов |
| 119 | Недопустимы номер возвращаемой строки таблицы идентификаторов |
| 120 | Размер длины строки превышен(256) |
| 121 | Превышено максимальное значение целочисленного литерала |
| 122 | Превышен максимальный размер имени идентификатора |

## **Параметры лексического анализатора и режимы его работы**

Транслятор допускает использование параметра для управления работой лексического анализатора, а именно выводом таблицы лексем и таблицы идентификаторов. Описание параметров представлено в таблице 2.2.

## **Алгоритм лексического анализа**

Лексический анализ является первой и наиболее простой фазой трансляции. Алгоритм лексического анализатора заключается в следующем: после разбиения текста из файла с исходным кодом на слова, для каждого слова подбирается конечный автомат, способный его разобрать, в случае если такой автомат существует, 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.

Работу конечных автоматов можно представить в виде графа. Пример графа представлен на рисунке 3.5. В виде кода представлен на рисунке 3.6. На рисунке 3.5 осуществляется разбор цепочки “byte”, где S0 — начальное состояние, а S4 — конечное.

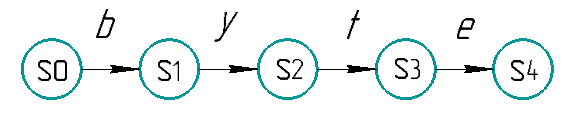


Рисунок 3.5 — Граф переходов для цепочки “byte”

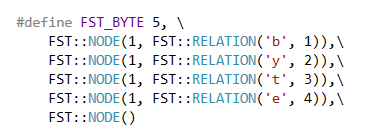


Рисунок 3.6 — Граф переходов для цепочки “byte”

## **Контрольный пример**

Результатом работы лексического анализатора являются таблица лексем и таблица идентификаторов. Содержимое таблиц на основе исходного кода из приложения А представлено в приложении В.

# **Глава 4 Разработка синтаксического анализатора**

## **Структура синтаксического анализатора**

Синтаксический анализатор является второй фазой работы транслятора. Назначением синтаксического анализаторы является распознавание синтаксических конструкций языка и формирование промежуточного кода. Исходными данными синтаксического анализатора являются таблицы лексем и идентификаторов. Свою работу начинает только при условии отсутствии ошибок в их общей структуре. Лексемы являются для синтаксического анализатора терминальными символами контекстно-свободной грамматики. Если в ходе его работы не было обнаружено ошибок, то формируется дерево разбора (промежуточное представление кода) и модифицируется протокол работы. Структура синтаксического анализатора представлена на рисунке 4.1.



Рисунок 4.1 — Структура синтаксического анализатора KAA-2019

## **Контекстно-свободная грамматика, описывающая синтаксис языка**

Каждый язык программирования описывается с помощью набора правил, определяющих структуру правильной программы. Наиболее удобным формализмом для описания синтаксических конструкций языка программирования являются контекстно-свободные грамматики.

Грамматика для синтаксического разбора языка KAA-2019 представляется четверкой G = <T, N, P, S>, где Т — множество терминальных символов, N — множество нетерминальных символов, P — множество правил языка, S — начальный символ грамматики, являющийся нетерминалом.

В грамматике языка KAA-2019 множество нетерминальных символов представлено следующим образом:

* S — Порождает правила, описывающие общую структуру программы;
* N — Порождает правила, описывающие инструкции языка
* Q — Порождает правила, описывающие формальные параметры функции;
* O — Порождает правила, описывающие параметры в стандартных функциях;
* Y — Порождает правила, описывающие выражения;
* E — Порождает правила, описывающие выражения;
* P — Порождает правила, описывающие арифметические выражения;
* M — Порождает правила, описывающие арифметические операции;
* B — Порождает правила, описывающие операции сравнения;
* K — Порождает правила, описывающие выражения в условном операторе;

Перечень правил, описывавших контекстно-свободную грамматику языка KAA-2019, представлен в таблице 4.1.

Таблица 4.1 – Правила грамматики KAA-2019

|  |  |
| --- | --- |
| Нетерминальный символ | Цепочки правил |
| S→ | tfi(Q)[Y]S | d[N] |
| N→ | ti:E.N | ti:E. | tO.N | tO. | ti:P.N | ti:P. | i:P.N | i:P. | i:E.N | i:E. | w(K)[N]e[N]N | w(K)[N]N | w(K)[N]e[N] | w(K)[N] | p(i).N | p(i). | p(l).N | p(l). |
| Q→ | ti | ti,Q | |
| O→ | i | i,O |
| Y→ | ti:E.Y | ti:E. | tO.Y | tO. | ti:P.Y | ti:P. | i:P.Y | i:P. | i:E.Y | i:E. | w(K)[Y]e[Y]Y | w(K)[Y]Y | w(K)[Y]e[Y] | w(K)[Y] | p(i).Y | p(i). | p(l).Y | p(l). | ri. | rli |
| E→ | i | l | s(i) | s(i,i) | s(l) | s(l,l) | s(O) | ~i | ~l |
| P→ | i | l | iP | lP | iMP | lMP | (P) | (P)MP | (P)N | (P) |
| M→ | + | - | \* | / | & | | | ~ |
| B→ | > | < | = |
| K→ | l | i | iBi | iBl | lBi | lBl |

## **Построение конечного магазинного автомата**

Конечный автомат с магазинной памятью представляет собой семерку,где Q — множество состояний автомата, V — алфавит входных символов, Z – алфавит специальных магазинных символов,  — функция переходов автомата,  — начальное состояние автомата,  — начальное состояние магазинного автомата, F — множество конечных состояний.

Пример работы магазинного автомата для цепочки представлен на рисунке 4.2.

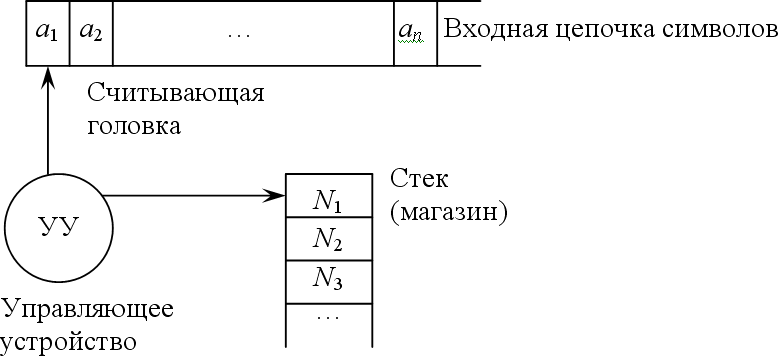


Рисунок 4.2 — Автомат с магазинной памятью

## **Основные структуры данных**

Основные структуры данных синтаксического анализатора включают в себя структуру магазинного автомата и структуру грамматики Грейбах, описывающая правила языка KAA-2019. Структура синтаксического анализатора представлена в приложении Г.

## **Описание алгоритма синтаксического разбора**

Входные символы и лексемы в форме Грейбах находятся в ленте на входе конечного автомата.

1) Если лента не пустая, переходим далее следующему пункту, иначе переходим к пункту 5.

2) Если на верхушке магазина нетерминальный символ.

2.1) Если есть такое правило, то переходим к следующему пункту.

2.1.1) Если цепочка есть, возвращаем NS\_OK. Переходим к пункту 4.

2.1.2) Иначе восстанавливаем состояние. Переходим к пункту 4.

2.2) Иначе возвращаем ошибку. Переход к пункту 4.

3) Если на верхушке терминал и он совпадает с символом на ленте, то удаляем его из стека и продвигаем ленту. Переход к пункту 4.

4) Повторяем шаг, переходим к пункту 1.

5) Конец работы.

## **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 |
| 600 | Неверная структура программы |
| 601 | Ошибка в написании инструкций или конструкций языка |
| 602 | Ошибка инициализации |
| 603 | Ошибка выражения |
| 604 | Ошибка знака в выражении |
| 605 | Ошибка знака в сравнении |
| 606 | Ошибка в условии if |
| 607 | Ошибка объявления |
| 608 | Ошибка параметров функции |
| 609 | Ошибки в теле функции |

## **Параметры синтаксического анализатора и режимы его работы**

Транслятор допускает использование параметра для управления работой синтаксического анализатора, а именно вывода дерева разбора и трассировки. Описание параметров представлено в таблице 2.2.

## **Принцип обработки ошибок**

Принцип заключается в том, что синтаксический анализатор перебирает все возможные правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем. В случае если не была найдена ни одна подходящая цепочка, то формируется соответствующая ошибка из таблицы 4.2. Все ошибки записываются в общую структуру ошибок, а также отображаются на консоли.

## **Контрольный пример**

Из контрольного примера, представленного в приложении А, результатом работы синтаксического анализатора, является трассировка и дерево разбора, представленные в приложении Д.

# **Глава 5 Разработка семантического анализатора**

## **Структура семантического анализатора**

Семантический анализатор в трансляторе языка KAA-2019 выделен в отдельную фазу. Проверка на ошибки в исходном коде производится после этапов лексического и синтаксического анализа. Общая структура обособленно работающего (не параллельно с лексическим анализом) семантического анализатора представлена на рисунке 5.1.

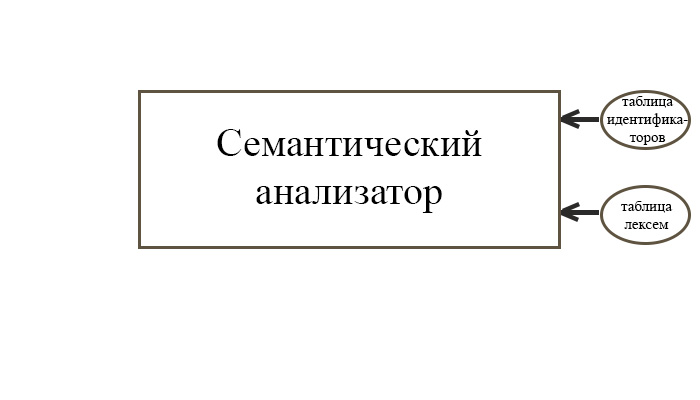


Рисунок 5.1 Структура семантического анализатора

## **5.2 Структура и перечень сообщений семантического анализатора**

Сообщения, формируемые семантическим анализатором представлены в таблице 5.1.

Таблица 5.1 — Перечень сообщений семантического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 700 | Тип данных передаваемых в функцию должен соответствовать заявленному. |
| 701 | Количество параметров передаваемых в функцию должно соответствовать заявленному значению. |
| 702 | Тип данных справа в выражении должен совпадать с типом слева |
| 703 | Для сравнения в блоке if без знаков сравнения могут использоваться только литералы и идентификаторы типа bool |
| 704 | Деление в выражении на ноль. |
| 705 | В программе может быть только одна точка входа |
| 706 | Применение знаков арифметических операций к идентификаторам или литералам типа string или bool |
| 707 | Применение знаков логических операций к идентификаторам или литералам типа string или bool |

Продолжение таблицы 5.1

|  |  |
| --- | --- |
| 708 | Тип данных возвращаемого значения не соответствует типу данных функции |
| 709 | Повторное объявление переменной |
| 710 | Использование необъявленной переменной |
| 711 | Неверно введено восьмеричное представление числа |

## **5.3 Принцип обработки ошибок**

Принцип обработки ошибок идентичен принципу обработки ошибок на этапе лексического анализа (раздел 3.6).

# 

# **Глава 6 Вычисление выражений**

## **Выражения, допускаемые языком**

Выражения, допускаемые языком KAA-2019, выполняются над целочисленными типами данных. В выражениях поддерживаются арифметические операции, такие как +, -, \*, / , &, | и (), и вызовы функций из них.

Приоритетность арифметических операций представлена в таблице 6.1.

Таблица 6.1 — Приоритетность операций

|  |  |  |
| --- | --- | --- |
| Операция | Приоритет | Порядок выполнения |
| + | 2 | Наименьший приоритет |
| - | 2 | Наименьший приоритет |
| | | 2 | Наименьший приоритет |
| & | 3 | Вторая степень приоритетности |
| \* | 3 | Вторая степень приоритетности |
| / | 3 | Вторая степень приоритетности |
| ( | 0 | Наивысший приоритет |
| ) | 0 | Наивысший приоритет |

## **Польская запись и принцип ее 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.

Обратная польская запись — это форма записи математических выражений, в которой операторы расположены после своих операндов. Выражение в обратной польской нотации читается слева направо: операция выполняется над двумя операндами, непосредственно стоящими перед знаком этой операции. Результат операции заменяет в выражении последовательность её операндов и символ операции. Результатом вычисления всего выражения является результат последней вычисленной операции.

Алгоритм построения польской записи:

* исходная строка: выражение;
* результирующая строка: польская запись;
* стек: пустой;
* исходная строка просматривается слева направо;
* операнды переносятся в результирующую строку;
* операция записывается в стек, если стек пуст;
* операция выталкивает все операции с большим или равным приоритетом в результирующую строку;
* отрывающая скобка помещается в стек;
* закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются.
* по концу разбора исходной строки все операции, оставшиеся в стеке, выталкиваются в результирующую строку.

## **Программная реализация обработки выражений**

Программная реализация алгоритма преобразования выражений к польской записи представлена в приложении Д.

## **Контрольный пример**

Результаты преобразования выражений исходного кода в польскую запись представлены в приложении Е. Ход разбора одного из выражений в контрольном примере в польский формат записи приведен в таблице 6.2.

Таблица 6.2 — Пример конвертации выражения в польскую запись

|  |  |  |
| --- | --- | --- |
| Выражение | Стек | Результат |
| ((45 + 25) / 2) \* (y - 2) & 4 |  |  |
| (45 + 25) / 2) \* (y - 2) & 4 | ( |  |
| 45 + 25) / 2) \* (y - 2) & 4 | (( |  |
| + 25) / 2) \* (y - 2) & 4 | (( | 45 |
| 25) / 2) \* (y - 2) & 4 | ((+ | 45 |
| ) / 2) \* (y - 2) & 4 | ((+ | 45 25 |
| / 2) \* (y - 2) & 4 | ( | 45 25 + |
| 2) \* (y - 2) & 4 | (/ | 45 25 + |
| ) \* (y - 2) & 4 | (/ | 45 25 + 2 |
| \* (y - 2) & 4 |  | 45 25 + 2 / |
| (y - 2) & 4 | \* | 45 25 + 2 / |
| y - 2) & 4 | \*( | 45 25 + 2 / |
| - 2) & 4 | \*( | 45 25 + 2 / y |
| 2) & 4 | \*(- | 45 25 + 2 / y |

Продолжение таблицы 6.2

|  |  |  |
| --- | --- | --- |
| ) & 4 | \*(- | 45 25 + 2 / y 2 |
| & 4 |  | 45 25 + 2 / y 2 - \* |
| 4 | & | 45 25 + 2 / y 2 - \* |
|  | & | 45 25 + 2 / y 2 - \* 4 |
|  |  | 45 25 + 2 / y 2 - \* 4 & |

# **Глава 7 Генерация кода**

## **7.1 Структура генератора кода**

Генерация объектного кода — это перевод транслятором представления исходной программы на языке KAA-2019 в цепочку символов выходного языка JavaScript. На вход генератора подаются таблицы лексем и идентификаторов, а также исходный код, разбитый на слова, на основе которых генерируется файл с кодом на языке JavaScript, который впоследствии будет интерпретироваться браузером. Схематично генерация кода показана на рисунке 7.1.Генератор кода начинает свою работу только в том случае, если код на языке KAA-2019 прошёл предыдущие этапы без ошибок.

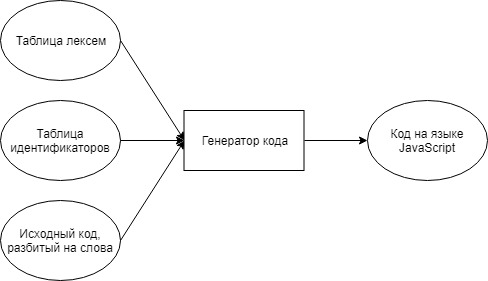


Рисунок 7.1 — Структура генератора кода

## **7.2 Представление типов данных в оперативной памяти**

Язык KAA-2019 требует указывать тип данных при объявлении идентификатора. Язык JavaScript не обязывает указывать тип, объявление происходит с указанием ключевого слова var и имени переменной. Соответствия между типами данных идентификаторов языка KAA-2019 и языка JavaScript представлены в таблице 7.1.

Таблица 7.1 — Соответствие типов данных

|  |  |
| --- | --- |
| Тип данных на языке KAA-2019 | Тип данных на языке JavaScript |
| byte (целочисленный) | Number (числовой) |
| string (строковый) | String (строковый) |
| bool (логический) | Boolean (логический) |

## **7.3 Статическая библиотека**

Для языка KAA-2019 статическая библиотека как отдельный файл не реализована. Преобразование функций происходит на этапе генерации кода.

## **7.4 Особенности алгоритма генерации кода**

Алгоритм генерации исходного кода на языке KAA-2019:

* открытие файла “Code.js” для записи;
* на вход генератора подаются таблицы лексем и идентификаторов, а также исходный код, разбитый на слова;
* каждый элемент таблицы лексем проверяется на соответствие с зарезервированными лексемами;
* если соответствие найдено, в выходной файл “Code.js” записывается соответствующее выражение. Пример данной операции для двух лексем представлен на рисунке 7.2;
* когда каждая лексема из таблицы разобрана, код сгенерирован, файл закрывается;

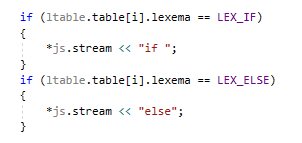


Рисунок 7.2 — Часть кода генерации

## **7.5 Входные параметры генератора кода**

На этапе генерации кода транслятор не допускает использование параметров. По умолчанию файлом для сгенерированного кода является файл “Code.js”. Проверку правильности генерации кода можно осуществить в любом html-файле, в который необходимо подключить сгенерированный код:

<script src="Code.js"></script>.

## **7.6 Контрольный пример**

Результат генерации кода на основе контрольного примера представлен в приложении Ж.

# **Глава 8 Тестирование транслятора**

## **8.1 Общие положения**

В результате обработки исходного кода программы, представленного в приложении А, транслятор языка KAA-2019 генерирует общий протокол работы, куда записываются все возникшие ошибки и предупреждения. Кроме того, все ошибки, возникшие на этапах лексического и семантического анализов, выводятся на консоль. Из ошибок, возникших на этапе синтаксического анализа, на консоль выводится только первая.

## **8.2 Результаты тестирования**

Транслятор языка KAA-2019 представляет диагностику и выявление ошибок на разных этапах трансляции. Ниже будут приведены результаты обработки транслятором исходного кода с допущенными ошибками.

Тестирование ошибок транслятора представлено в таблице 8.1. В таблице 8.1 приведены три вида ошибок, первая происходит на этапе лексического анализа, вторая — синтаксического, третья — семантического.

Таблица 8.1 — Тестирование

|  |  |
| --- | --- |
| Исходный код с ошибкой | Генерируемое сообщение об ошибке |
| program  [  byte a : 321.  print (a).  ] | Ошибка 121: Превышено максимальное значение целочисленного литерала  Строка: 2, символ: 13 |
| program  byte y : 11.  print (y).  ] | Ошибка 600: Неверная структура программы  Строка: 4, символ: 8 |
| program  [  byte c : "string".  print (c).  ] | Ошибка 702: Тип данных справа в выражении должен совпадать с типом слева  Строка: 3, символ: 16 |

# **Заключение**

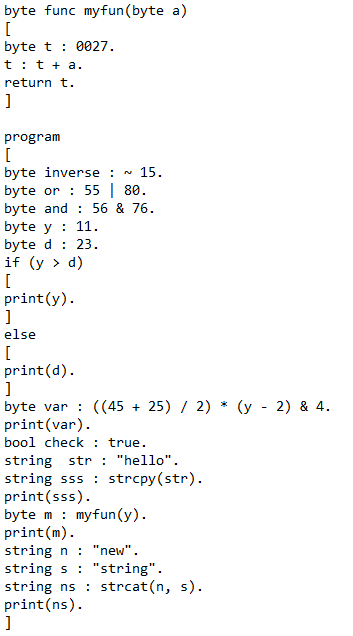
В ходе выполнения курсовой работы был разработан транслятор для языка программирования KAA-2019. Таким образом, были выполнены основные задачи данной курсовой работы:

* сформулирована спецификация языка KAA-2019;
* разработаны конечные автоматы и алгоритмы для реализация лексического анализатора;
* разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
* разработан семантический анализатор, осуществляющий проверку смысла используемых инструкций;
* разработан транслятор с языка программирования KAA-2019 на язык Ассемблера;
* проведено тестирование всех вышеперечисленных компонентов.

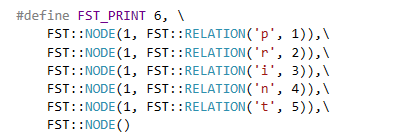
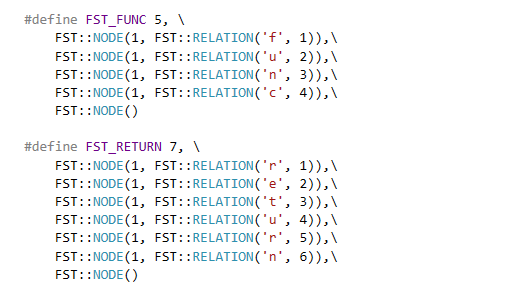
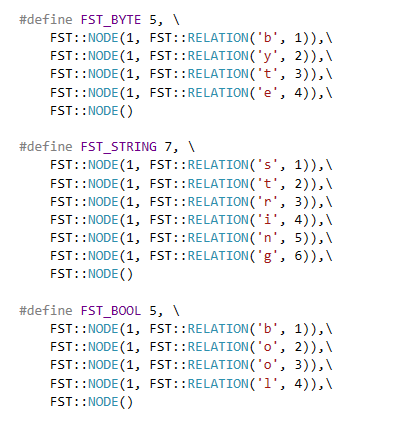
Окончательная версия языка KAA-2019 включает:

* 3 типа данных;
* Поддержка операции вывода;
* Возможность вызова функций стандартной библиотеки;
* Наличие 4 арифметических операторов и 3 побитовых для вычисления выражений;
* Наличие условного оператора;
* Структурированная система для обработки ошибок пользователя.

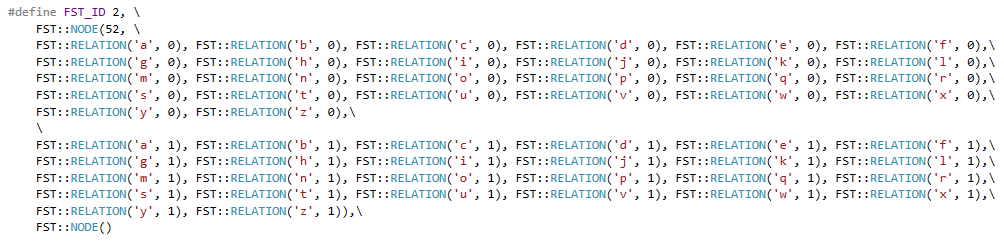
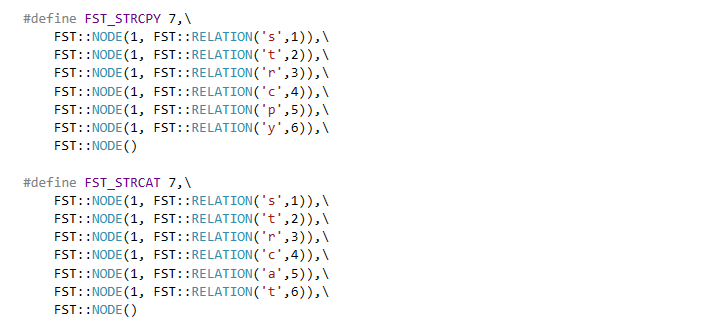
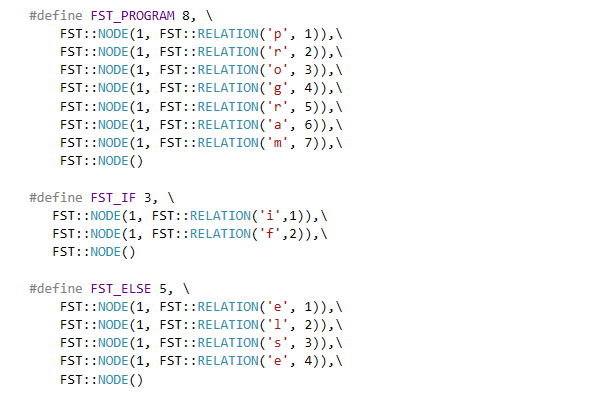
# **Приложение А**



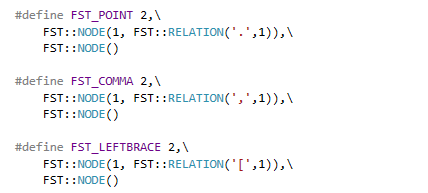
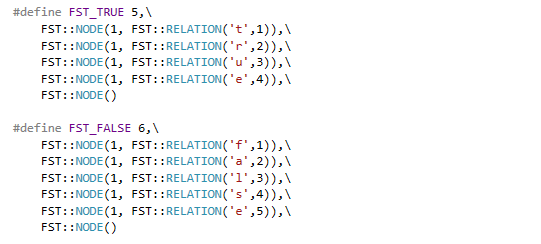
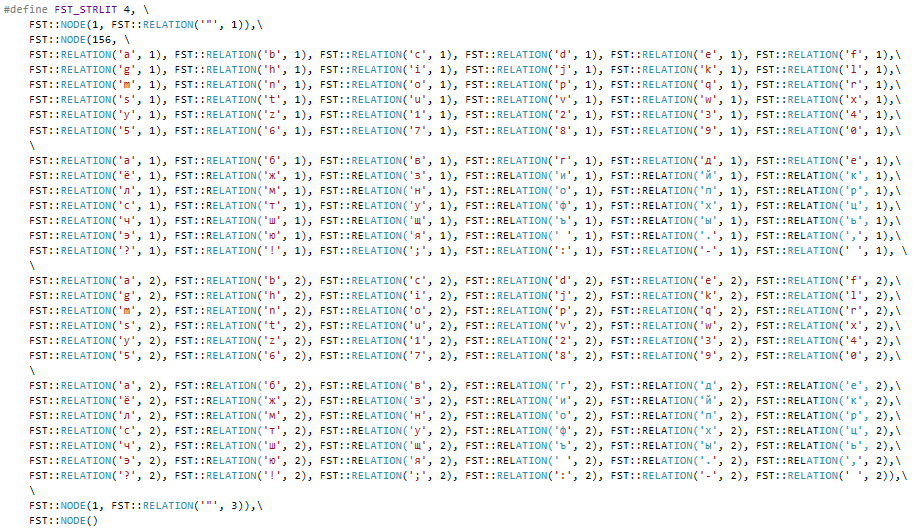
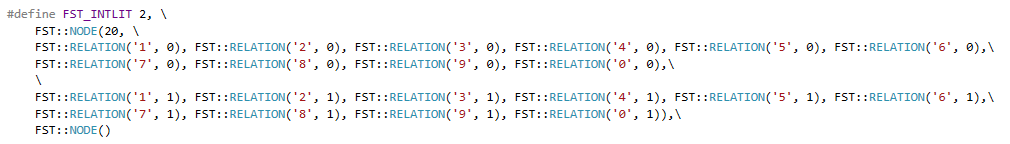
# **Приложение Б**



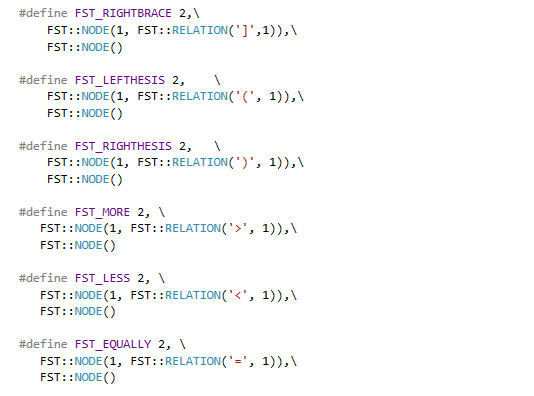
Продолжение приложения Б



Продолжение приложения Б

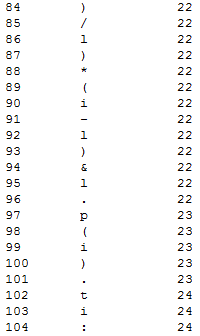
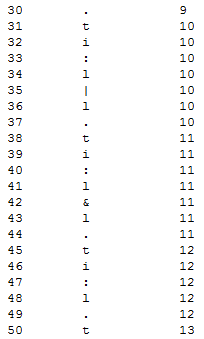
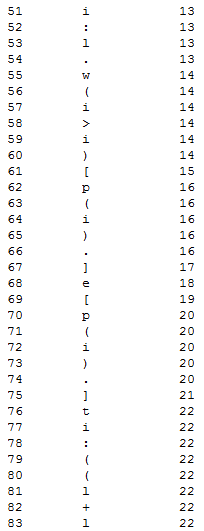
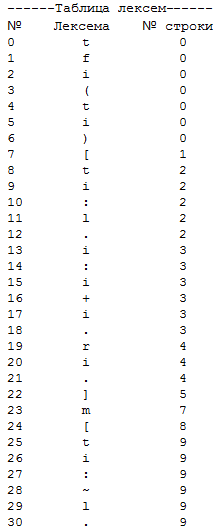


Продолжение приложения Б

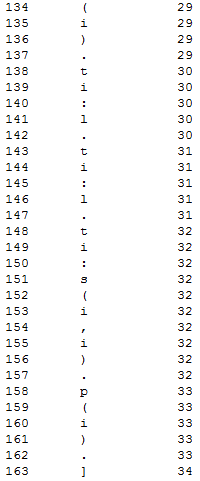
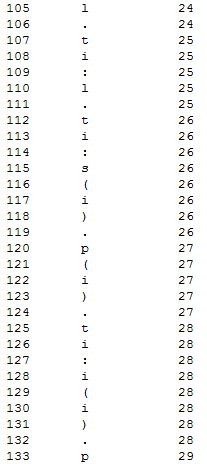


C:\Users\Александр\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Screenshot_47.png

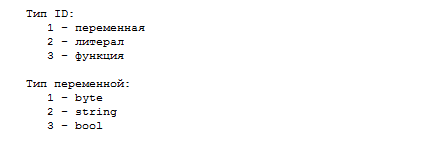
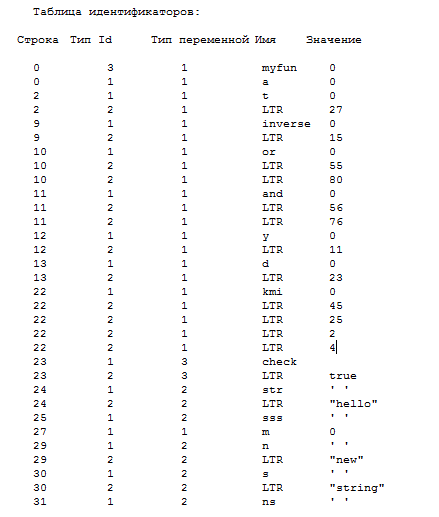
**Приложение В**



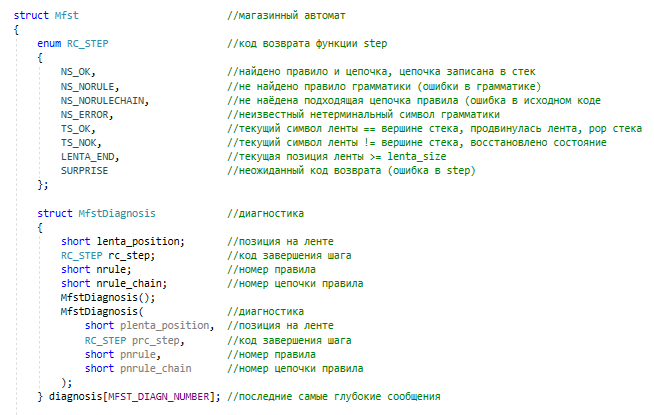
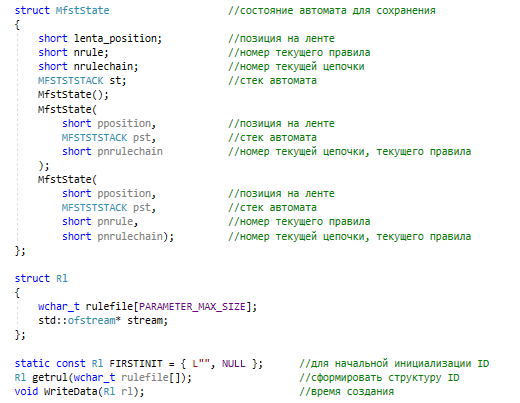
Продолжение приложения B



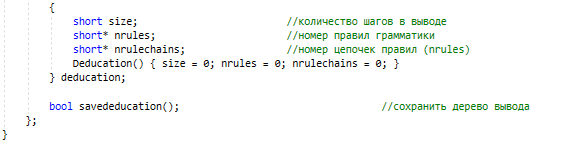
Продолжение приложения В



# **Приложение Г**

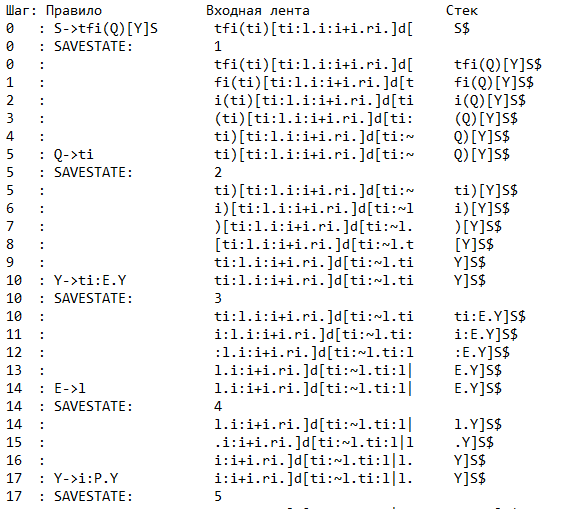


Продолжение приложения Г

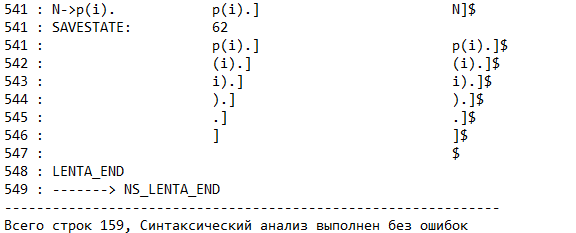


# **Приложение Д**

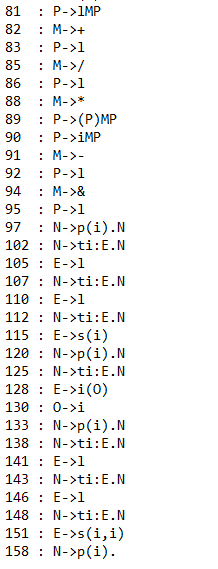
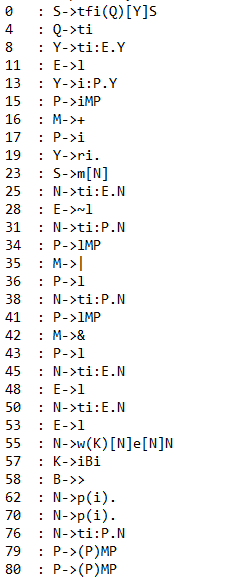
Начало разбора



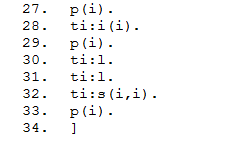
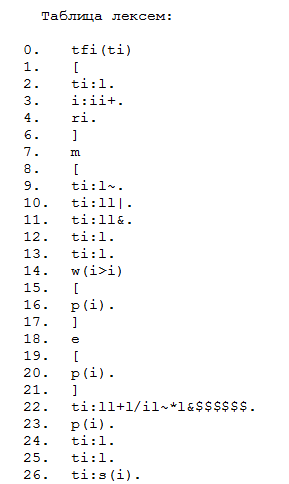
Конец разбора



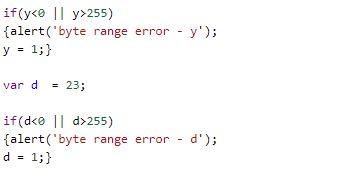
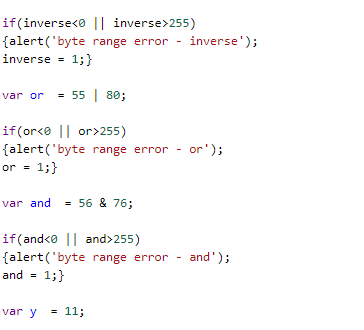
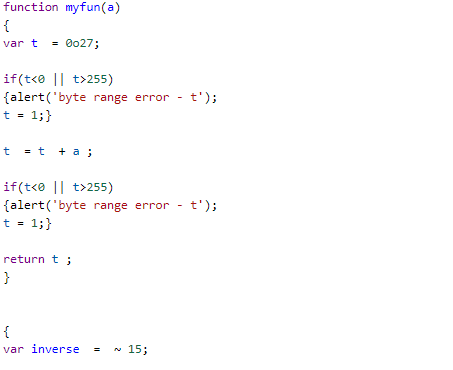
Продолжение приложения Д



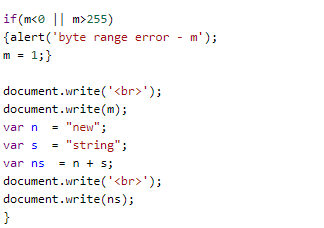
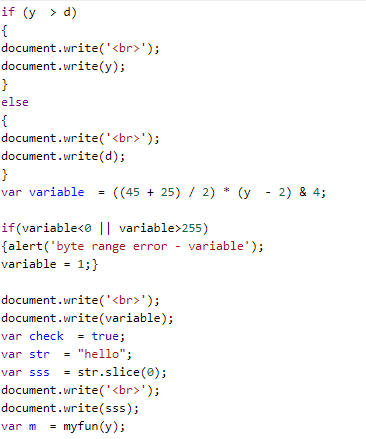
# **Приложение Е**



# **Приложение Ж**



Продолжение приложения Ж



# **Литература**

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

2. Молчанов, А. Ю. Системное программное обеспечение / А. Ю. Молчанов. – СПб.: Питер, 2010. – 400 с.

3 Разработка компиляторов / А.А. Терехов, А.Е. Москаль, Д.Ю. Булычев, Н.Н. Вояковская, 2016. – 375с.

4. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. - 3-е изд. – Москва : Вильямс, 2003. - 429 с.

5. Орлов, С.А. Теория и практика языков программирования / С.А. Орлов – 2014. – 689 с.

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