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 KAV-2020»

Выполнил студент Качанова Анастасия Васильевна

(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.)

Нормоконтролер ст.пр. Наркевич Аделина Сергеевна

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

Курсовой проект защищен с оценкой

Минск 2020

[Введение 4](#_Toc58778997)

[1. Спецификация языка программирования 5](#_Toc58778998)

[1.1 Характеристика языка программирования 5](#_Toc58778999)

[1.2 Определение алфавита языка программирования 5](#_Toc58779000)

[1.3 Применяемые сепараторы 5](#_Toc58779001)

[1.4 Применяемые кодировки 6](#_Toc58779002)

[1.5 Типы данных 6](#_Toc58779003)

[1.6 Преобразование типов данных 7](#_Toc58779004)

[1.7 Идентификаторы 7](#_Toc58779005)

[1.8 Литералы 7](#_Toc58779006)

[1.9 Объявление данных 8](#_Toc58779007)

[1.10 Инициализация данных 8](#_Toc58779008)

[1.11 Инструкции языка 9](#_Toc58779009)

[1.12 Операция языка 9](#_Toc58779010)

[1.13. Выражения и их вычисление 10](#_Toc58779011)

[1.14. Конструкции языка 10](#_Toc58779012)

[1.15 Область видимости идентификаторов 11](#_Toc58779013)

[1.16 Семантические проверки 11](#_Toc58779014)

[1.17 Распределение оперативной памяти на этапе выполнения 12](#_Toc58779015)

[1.18 Стандартная библиотека и ее состав 12](#_Toc58779016)

[1.19 Ввод и вывод данных 12](#_Toc58779017)

[1.20 Точка входа 12](#_Toc58779018)

[1.21 Препроцессор 12](#_Toc58779019)

[1.22 Соглашения о вызовах 12](#_Toc58779020)

[1.23 Объектный код 13](#_Toc58779021)

[1.24 Классификация сообщений транслятора 13](#_Toc58779022)

[1.25 Контрольный пример 13](#_Toc58779023)

[2. Структура транслятора 14](#_Toc58779024)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 14](#_Toc58779025)

[2.2 Перечень входных параметров транслятора 15](#_Toc58779026)

[2.3 Протоколы, формируемые транслятором 15](#_Toc58779027)

[3. Разработка лексического анализатора 16](#_Toc58779028)

[3.1 Структура лексического анализатора 16](#_Toc58779029)

[3.2 Контроль входных символов 16](#_Toc58779030)

[3.3 Удаление избыточных символов 17](#_Toc58779031)

[3.4 Перечень ключевых слов 17](#_Toc58779032)

[3.5. Основные структуры данных 17](#_Toc58779033)

[3.6 Структура и перечень сообщений лексического анализатора 19](#_Toc58779034)

[3.7 Принцип обработки ошибок 19](#_Toc58779035)

[3.8 Параметры лексического анализатора 19](#_Toc58779036)

[3.9 Алгоритм лексического анализатора 19](#_Toc58779037)

[3.10 Контрольный пример 20](#_Toc58779038)

[4. Разработка синтаксического анализатора 21](#_Toc58779039)

[4.1 Структура синтаксического анализатора 21](#_Toc58779040)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 21](#_Toc58779041)

[4.3 Построение конечного магазинного автомата 23](#_Toc58779042)

[4.4 Основные структуры данных 24](#_Toc58779043)

[4.5 Описание алгоритма синтаксического разбора 24](#_Toc58779044)

[4.6 Структура и перечень сообщений синтаксического анализатора 24](#_Toc58779045)

[4.7 Параметры синтаксического анализатора и режимы его работы 24](#_Toc58779046)

[4.8 Принцип обработки ошибок 25](#_Toc58779047)

[4.9 Контрольный пример 25](#_Toc58779048)

[5. Разработка семантического анализатора 26](#_Toc58779049)

[5.1 Структура семантического анализатора 26](#_Toc58779050)

[5.2 Функции семантического анализатора 26](#_Toc58779051)

[5.3 Структура и перечень сообщений семантического анализатора 26](#_Toc58779052)

[5.5 Контрольный пример 27](#_Toc58779053)

[6. Вычисление выражений 28](#_Toc58779054)

[6.1. Выражения, допускаемые языком 28](#_Toc58779055)

[6.2. Польская запись и принцип ее построения 28](#_Toc58779056)

[6.3. Программная реализация обработки выражений 29](#_Toc58779057)

[6.4. Контрольный пример 29](#_Toc58779058)

[7. Генерация кода 31](#_Toc58779059)

[7.1. Структура генератора кода 31](#_Toc58779060)

[7.2. Представление типов данных в оперативной памяти 31](#_Toc58779061)

[7.3. Статическая библиотека 32](#_Toc58779062)

[7.4. Особенности алгоритма генерации кода 32](#_Toc58779063)

[7.5. Входные параметры генератора кода 32](#_Toc58779064)

[7.6. Контрольный пример 32](#_Toc58779065)

[8. Тестирование транслятора 33](#_Toc58779066)

[8.1. Общие положения 33](#_Toc58779067)

[8.2. Результаты тестирования 33](#_Toc58779068)

[Заключение 34](#_Toc58779069)

[Список использованных источников 35](#_Toc58779070)

[Приложения A 36](#_Toc58779071)

[Приложение Б 37](#_Toc58779072)

[Приложение В 43](#_Toc58779073)

[Приложение Г 46](#_Toc58779074)

[Приложение Д 48](#_Toc58779075)

[Приложение Е 50](#_Toc58779076)

[Приложение Ж 51](#_Toc58779077)

# **Введение**

Основной целью данного курсового проекта является разработка транслятора для языка программирования KAV-2020. В данном курсовом проекте трансляция будет осуществляться в код на языке JavaScript.

Исходя из цели курсового проекта, были определены следующие задачи:

– разбработка спецификации языка программирования;

– разработка структуры транслятора;

– разработка лексического анализатора;

– разработка синтаксического анализатора;

– разработка семантического анализатора;

– обработка выражений;

– генерация кода на язык JavaScript;

– тестирование транслятора.

Решения каждой из поставленных задач будут 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.

Язык программирования KAV-2020 предназначен для работы с консолью, выполняя простейшие логические операции и арифметические действия.

# **1. Спецификация языка программирования**

## **1.1 Характеристика языка программирования**

Язык программирования KAV-2020 – это универсальный язык высокого уровня. Он является процедурным, компилируемым, не объектно-ориентированным, строго типизируемым.

## **1.2 Определение алфавита языка программирования**

Исходный код KAV-2020 может содержать символы латинского алфавита малого и верхнего регистра, цифры десятичной системы счисления от 0 до 9, русские символы разрешены только в строковых литералах.

## **1.3 Применяемые сепараторы**

Сепараторы нужны для разделения операций языка. Символы, которые являются сепараторами представлены в таблице 1.1.

Таблица 1.1 – Символы сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение |
| ;  = | Разделитель программных конструкций |
| ' ' (пробел) | Разделитель цепочек. Допускается везде, кроме названий идентификаторов и ключевых слов |
| , | Разделитель параметров функций |
| >  <  & (больше или равно)  $(меньше или равно)  ! (неравнство)  ? (равенство) | Логические операции (Сравнения: больше, меньше, равенство, неравенство, больше или равно, меньше или равно), используемые в условии цикла |
| + - \* / | Арифметические операции |
| = | Оператор присваивания |
| {…} | Программный блок инструкций |
| (…) | Параметры функций / приоритетность операций (в выражениях) |

## **1.4 Применяемые кодировки**

Для написания исходного кода на языке программирования KAV-2020 используется кодировка Windows-1251, представленная на рисунке 1.1

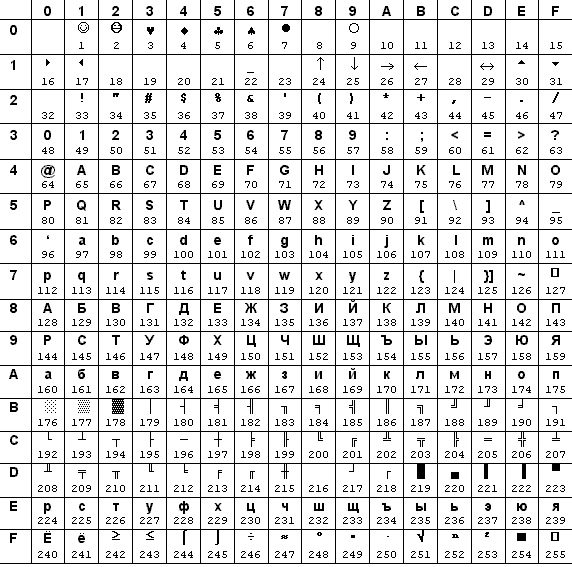


Рисунок 1.1 Кодировка Windows 1251

**1.5 Типы данных**

В языке KAV-2020 используется три типа данных: беззнаковый целочисленный, символьный и строковый. Пользовательские типы данных не поддерживаются. Описание типов данных, предусмотренных в данным языке представлено в таблице 1.2.

Таблица 1.2 – Типы данный языка KAV-2020

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| int | Фундаментальный тип данных. Используется для работы с числовыми значениями. Инициализация по умолчанию: значение 0. Поддерживаемые операции:  **+** (бинарный) – оператор сложения;  - (бинарный) – оператор вычитания;  **\*** (бинарный) – оператор умножения;  **/** (бинарный) – оператор деления;  **=** (бинарный) – оператор присваивания.  В качестве условия условного оператора поддерживаются следующие логические операции:  **>** (бинарный) – оператор «больше»;  **<** (бинарный) – оператор «меньше»; |

Продолжение таблицы 1.2

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| char | Фундаментальный тип данных. Предусмотрен для объявления символов. Автоматическая инициализация нулевым значением. |
| string | Фундаментальный тип данных. Используется для работы с символами, каждый символ в памяти занимает 1 байт. Максимальное количество символов: 255.  Инициализация по умолчанию: длина 0, символ конца строки “\0”.  Операции над данными строкового типа: присваивание строковому идентификатору значения другого строкового идентификатора, строкового литерала или значения строковой функции, а также использование библиотечных функций. |

**1.6 Преобразование типов данных**

В языке программирования KAV-2020 преобразование типов данных не поддерживается, все типы данных определены однозначно.

## **1.7 Идентификаторы**

Идентификатор — имя компонента программы (переменной или функции), составленное программистом по определенным правилам. Идентификаторы используются для именования функций, параметров и переменных. В языке KAV-2020 не предусмотрены зарезервированные идентификаторы. Идентификаторы должны содержать только символы верхнего и нижнего регистров латинского алфавита.

Примеры правильных идентификаторов: Now, This, a и т.д.

Примеры неправильных идентификаторов: Chicken!, 1var, ab\_c и т.д.

## **1.8 Литералы**

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

В языке существует два типа литералов:

* целые
* строковые
* символьные

Правило записи литералов представлено в таблице 1.3.

Таблица 1.3 – Правило записи литералов

|  |  |
| --- | --- |
| Тип литерала | Описание |
| Числовые | Целочисленные неотрицательные литералы в десятичной или шестнадцатеричной системе счисления, инициализируются 0. Не имеют дробных частей, только положительные. |
| Символьные | Символы, заключённые в ' '(одинарные кавычки), инициализируются пустой строкой, максимальное число символов 1. |
| Строковые | Символы, заключённые в ' '(одинарные кавычки), инициализируются пустой строкой, максимальное число символов 28-1=255. |

Ограничения на строковые литералы языка KAV-2020: внутри литерала не допускается использование кириллицы, а также одинарных и двойных кавычек. Ограничения на целочисленные литералы: не могут начинаться с нуля, если их значение не ноль, если литерал отрицательный, после знака '-' не может быть нуля.

## **1.9 Объявление данных**

Областью видимости переменной является блок функции (область видимости «сверху вниз», по принципу С++), в которой она определена. В языке KAV-2020 требуется обязательное объявление переменной перед её использованием. Имеется возможность объявления одинаковых переменных в разных блоках. Не допустимо объявление глобальных переменных. Конструкция для объявления переменной:

let <тип\_даных> <идентификатор>;

Примеры: let int a, let string b;

Для объявления функций используется ключевое слово def, перед которым указывается тип возвращаемого значения функции, а после – имя функции либо процедуры. Далее обязателен список параметров и тело функции.

## **1.10 Инициализация данных**

Допускается инициализация данных при объявлении переменной. При этом переменной будет присвоено значение литерала или идентификатора, стоящего справа от знака равенства. Объектами-инициализаторами могут быть идентификаторы, литералы, выражения и вызовы функций. Предусмотрены значения по умолчанию, если переменные не инициализированы. Способы инициализации переменных представлены в таблице 1.4.

Таблица 1.4 – Способы инициализации переменных

|  |  |
| --- | --- |
| Вид инициализации | Примечание |
| let <тип\_данных> <идентификатор>; | Автоматическая инициализация: int инициализируются нулём, char – пустым символом, string – нулевым символом. |
| <идентификатор> = <значение>; | Присваивание переменной значения. |

## **1.11 Инструкции языка**

Все возможные инструкции языка программирования KAV-2020 представлены в общем виде в таблице 1.5.

Таблица 1.5 – Инструкции языка

|  |  |
| --- | --- |
| Инструкция | Синтаксис |
| Объявление переменной | let <тип данных> <идентификатор>; |
| Присваивание | <идентификатор>=<значение>|<идентификатор>|  <выражение>; |
| Главная функция | main  {  <инструкции языка>  }; |
| Вызов функции | <идентификатор функции>(<идентификатор>|  <литерал, <список параметров>); |
| Возврат функции | return <идентификатор>|<литерал>; |
| Объявление функции | <тип данных> def <идентификатор>  {  <инструкции языка>  }; |
| Вывод данных | print (<идентификатор> | <литерал> | <выражение>); |
| Условный оператор | if(<условие>){…}  else{…} |

## **1.12 Операция языка**

Операции сравнения применимы к беззнаковым целым и строковым типам данных. Арифметические операции применяются к беззнаковым целым. Наибольшую приоритетность арифметических операций имеют операции умножения и деления, а сложение и вычитание меньшую. При одинаковом приоритете первой выполнится операция, расположенная левее. Изменить приоритетность можно с помощью круглых скобок.

Язык программирования KAV-2020 может выполнять операции, представленные в таблице 1.6.

Таблица 1.6 – операции языка программирования KAV-2020

|  |  |
| --- | --- |
| Тип операции | Оператор |
| Арифметические | () – приоритетность операции  + – сложение  - – вычитание  \* – умножение  / – деление |
| Логические | ? – равенство  ! – неравенство  > – больше  < – меньше  & – больше или равно  $ – меньше или равно |

## **1.13. Выражения и их вычисление**

Выражением называется совокупность переменных, знаков операций, имён функций, скобок, которая может быть вычислена в соответствии с синтаксисом языка программирования. Всякое выражение составляется согласно следующим правилам:

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

## **1.14. Конструкции языка**

Программа на языке KAV-2020 оформляется в виде функций пользователя и главной функции. При составлении функций рекомендуется выделять блоки и фрагменты и применять отступы для лучшей читаемости кода.

Ключевые программные конструкции языка программирования KAV-2020 представлены в таблице 1.7.

Таблица 1.7 – программные конструкции языка

|  |  |
| --- | --- |
| Название | Синтаксис |
| Главная функция (точка входа в приложение) | main  {…}; |

Продолжение таблицы 1.7

|  |  |
| --- | --- |
| Название | Синтаксис |
| Функция | <тип данных> def <идентификатор>(<тип данных> <идентификатор>, …)  { …  return <выражение>;  }; |
| Условный оператор | if(<условие>){ … }  else{ … }; |

## **1.15 Область видимости идентификаторов**

В языке KAV-2020 все идентификаторы должны находится внутри программного блока функций. Глобальных переменных нет, только локальные. Параметры видны только внутри функции, в которой объявлены. Область видимости используется по принципу сверху вниз.

## **1.16 Семантические проверки**

Таблица с перечнем семантических проверок, предусмотренных языком, приведена в таблице 1.8.

Таблица 1.8 – семантические проверки

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Переопределение идентификаторов |
| 2 | Тип данных переменной должен совпадать с типом значения, которое присваивается этому типу |
| 3 | Идентификатор должен быть объявлен до его использования. |
| 4 | Операнды в арифметическом или логическом выражениях не могут быть разных типов |
| 5 | Тип данных возвращаемой функции должен совпадать с типом при ее объявлении |
| 6 | Наличие функции main, как точки входа в программу |
| 7 | Наличие только одной точки входа |
| 8 | Правильность передаваемых в функцию параметров: количество, типы |
| 9 | Деление на ноль запрещено |

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

Все переменные размещаются в стеке. Таблица лексем и таблица идентификаторов сохраняются в структуры с выделенной под них динамической памятью, которая очищается по окончанию работы транслятора.

## **1.18 Стандартная библиотека и ее состав**

Функции стандартной библиотеки с описанием представлены в таблице 1.9.

Таблица 1.9 – Cостав стандартной библиотеки

|  |  |  |  |
| --- | --- | --- | --- |
| Имя функции | Возвращаемое значение | Принимаемые параметры | Описание |
| root | int | int a – число | Извлекает корень числа a |
| power | int | int x , int y | Возводит число x в степень y |
| src | string | string a , string b | Совершает конкатенацию строк |
| srt | string | string a , string b | Копирует значение одной строки в другую и наоборот. |

## **1.19 Ввод и вывод данных**

Ввод данных не поддерживается языком программирования KAV-2020. Вывод данных осуществляется программной конструкцией print, в скобках указывается имя идентификатора или литерала. Вычисление выражений внутри функции не предусматривается.

print (<идентификатор или литерал>); – вывод в стандартный поток вывода.

## **1.20 Точка входа**

Точкой входа является функция main. В программе может быть только одна точка входа

## **1.21 Препроцессор**

Не предусмотрено подключение пользовательских библиотек.

**1.22 Соглашения о вызовах**

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

В языке KAV-2020 по умолчанию применяется соглашение \_stdcall, где параметры помещаются в стек, передача параметров происходит справа налево, стек освобождает вызываемый код, возврат через регистр EAX.

## **1.23 Объектный код**

Исходный код, написанный на языке KAV-2020, транслируется нашим компилятором в язык JavaScript, после чего интерпретируется браузером.

**1.24 Классификация сообщений транслятора**

В случае возникновения ошибки в коде программы на языке KAV-2020 и выявления её транслятором в текущий файл протокола выводится сообщение. Их классификация сообщений приведена в таблице 1.10.

Таблица 1.10 – Классификация сообщений транслятора

|  |  |
| --- | --- |
| Интервал | Описание ошибок |
| 0 – 9 | Системные ошибки |
| 10 – 19 | Ошибки входных параметров |
| 20 – 29 | Ошибки открытия и чтения файлов |
| 30 – 49 | Ошибки лексического анализа |
| 50 – 79 | Ошибки синтаксического анализа |
| 80 – 99 | Ошибки семантического анализа |

**1.25 Контрольный пример**

Контрольный пример демонстрирует главные особенности языка KAV-2020: его фундаментальные типы, основные структуры, функции. Исходный код контрольного примера представлен в приложении А.

# **2. Структура транслятора**

**2.1 Компоненты транслятора, их назначение и принципы взаимодействия**

Транслятор языка KAV-2020 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.

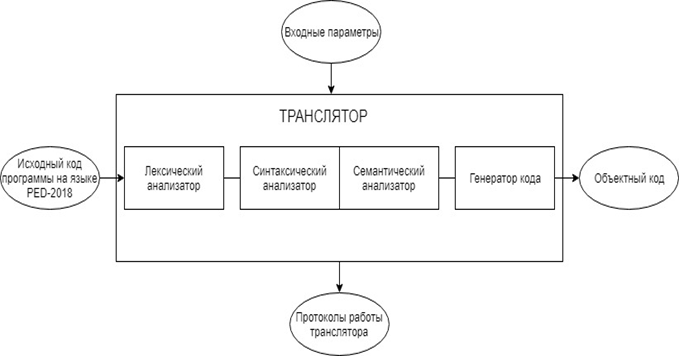


Рисунок 2.1. Схема структура транслятора

Лексический анализатор – часть транслятора, выполняющая лексический анализ. На данном этапе распознаётся правильность составления лексем и идентификаторов. На вход лексического анализатора подаётся последовательность символов входного языка. Лексический анализатор преобразует исходный текст, заменяя лексические единицы их внутренним представлением – лексемами, для создания промежуточного представления исходной программы. Каждой лексеме сопоставляется ее тип и запись в таблице идентификаторов, в которой хранится дополнительная информация. После окончания данного этапа формируется таблица лексем и таблица идентификаторов, которые являются входом для следующей фазы компилятора – синтаксического анализа.

Синтаксический анализатор – часть транслятора, выполняющая синтаксический анализ, то есть проверку исходного кода на соответствие правилам грамматики. Для того, чтобы провести данную операцию используются таблица лексем и идентификаторов. На выходе формируется дерево разбора.

Семантический анализатор – часть транслятора, выполняющая семантический анализ, то есть исходный код проверяется на наличие ошибок, которые невозможно отследить при помощи регулярной и контекстно-свободной грамматики. Входными данными являются таблица лексем и идентификаторов.

Генератор кода – часть транслятора, выполняющая генерацию кода на основе полученных данных на предыдущих этапах трансляции. На вход генератора подаются таблица лексем и таблица идентификаторов, на основе которых генерируется файл с кодом JavaScript.

## **2.2 Перечень входных параметров транслятора**

Перечень входных параметров транслятора представлен в таблице 2.1.

Таблица 2.1 – входные параметры транслятора

|  |  |
| --- | --- |
| Входной параметр | Описание |
| -in: | Указывает транслятору путь к исходному коду. Является обязательным параметром. В случае его отсутствия трансляция выполняться не будет. Имеет расширение .txt |
| -out: | Выходной файл – результат работы транслятора. |
| -log: | Указывает транслятору в какой файл выводить протокол работы транслятора. |

**2.3 Протоколы, формируемые транслятором**

Перечень протоколов, формируемых транслятором, представлен в таблице 2.2.

Таблица 2.2 – Протоколы, формируемые транслятором

|  |  |
| --- | --- |
| Протокол | Описание протокола |
| Выходной файл, заданный параметром “-out”: | Файл с протоколом работы транслятора языка программирования KAV-2020. Содержит таблицу лексем, результат работы алгоритма преобразования выражений к польской записи |
| Файл журнала, заданный параметром “-log:” | Результат работы программы. Выводится таблица лексем. Выводится таблица идентификаторов. Выводится правила, по которым осуществился разбор исходного кода. Выводится полная информация о разборе таблицы лексем на синтаксическом анализаторе. |

# **3. Разработка лексического анализатора**

**3.1 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 KAV-2020. В результате работы лексического анализатора формируется таблица идентификаторов и таблица лексем, модифицируется протокол работы транслятора. Структура лексического анализатора KAV-2020 представлена на рисунке 3.1.



Рисунок 3.1 Структура лексического анализатора

**3.2 Контроль входных символов**

Таблица для контроля входных символов представлена на рисунке 3.2.

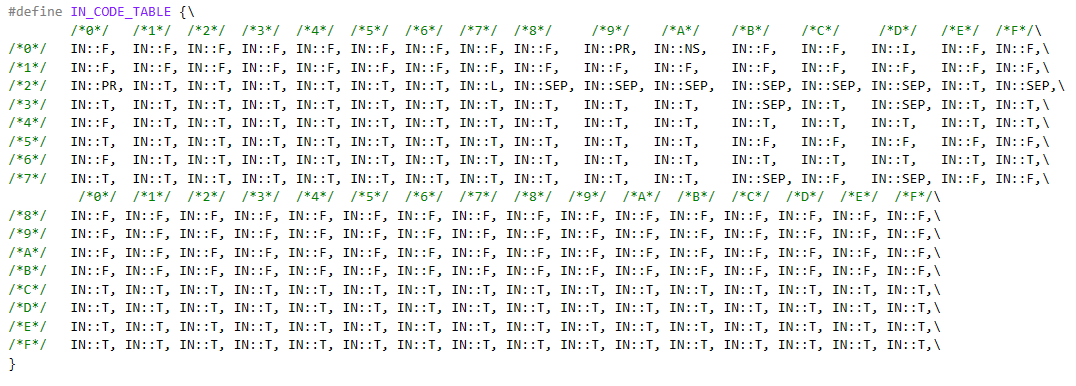


Рисунок 3.2 Таблица для контроля входных символов

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, PR — пробел, , SEP — символы-сепараторы,

NS — новая строка, L — кавычка

**3.3 Удаление избыточных символов**

Избыточными в языке программирования KAV-2020 считаются символы пробела и табуляции, не влияющие на ход выполнения программы. Избыточные символы удаляются на этапе разбиения исходного кода на слова.

Описание алгоритма удаления избыточных символов:

1. Файл с исходным кодом программы считывается посимвольно;
2. В конце каждой итерации посимвольного считывания текущий символ сохраняется в буфер;
3. встреча пробела включает проверку предыдущего (буферного) символа;
4. если предыдущий символ является допустимым символом, текущий пробел — разделитель единиц языка, текущий символ не игнорируется;
5. если предыдущий символ является пробелом, текущий символ игнорируется;
6. если текущий символ — кавычка, отключается игнорирование пробелов до тех пор, пока не встретится еще одна кавычка (т.к. внутри кавычек содержатся данные пользователя — литералы);

**3.4 Перечень ключевых слов**

Ключевые слова языка KAV-2020: let, int, string, def, return, main, print, EMathLib, SLib, power, root, src, srt.

Для каждого слова, операторов и сепараторов определен конечный автомат, по которому происходит разбор выражения. 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.

Благодаря замене цепочек, написанных на языке KAV-2020, лексемами, упрощается дальнейшая обработка исходного кода программы. Перечень цепочек, соответствующих им автоматов и лексем, представлена в приложении Б.

**3.5. Основные структуры данных**

Основными структурами данных лексического анализатора являются таблица лексем и таблица идентификаторов. Таблица лексем содержит номер лексемы, лексему, полученную при разборе, номер строки в исходном коде и номер в таблице идентификаторов, если лексема является идентификатором. Таблица идентификаторов содержит имя идентификатора, номер в таблице лексем, тип данных, тип идентификатора, значение (или параметры функций), тип литерала, область видимости.

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

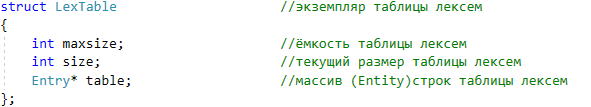
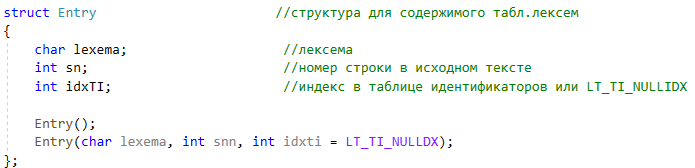
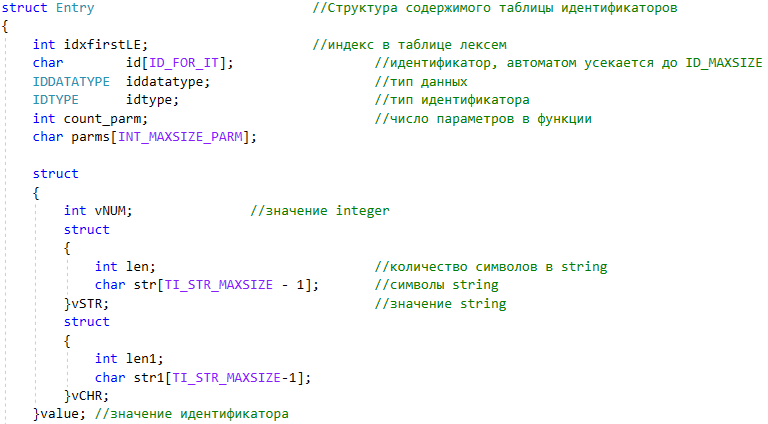


Рисунок 3.3 Структуры таблиц лексем KAV-2020

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



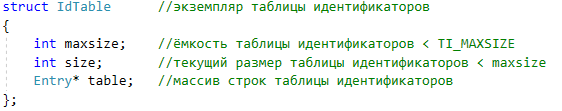


Рисунок 3.4 Структуры таблиц идентификаторов KAV-2020

**3.6 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 |
| 30 | Превышен размер таблицы идентификаторов |
| 31 | Превышен размер таблицы лексем |
| 32 | Слово не разобрано |

**3.7 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 не прекращается, что даёт возможность анализировать одновременно несколько возникших проблем.

**3.8 Параметры лексического анализатора**

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

**3.9 Алгоритм лексического анализатора**

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 “main”, где S0 — начальное состояние, а S4 — конечное.



Рисунок 3.5 — Граф переходов для цепочки “main”

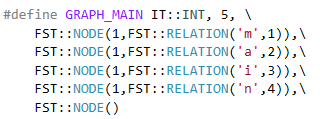


Рисунок 3.6 — Граф переходов для цепочки “main”

**3.10 Контрольный пример**

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

**4. Разработка синтаксического анализатора**

**4.1 Структура синтаксического анализатора**

Синтаксический анализатор – часть транслятора, выполняющая синтаксический анализ. На данном этапе исходный код проверяется на соответствие правилам грамматики. Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов. Лексемы являются для синтаксического анализатора терминальными символами контекстно-свободной грамматики. На выходе получается дерево разбора и протокол работы анализатора. Структура синтаксического анализатора представлена на рисунке 4.1.



Рисунок 4.1 Структура синтаксического анализатора

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

Синтаксис языка KAV-2020 описывается при помощи контекстно свободной грамматики.

Контекстно-свободная грамматика – грамматика типа 2 по иерархии Хомского. Данная грамматика имеет вид , где

T – множество терминальных символов,

N – множество нетерминальных символов,

P – множество правил языка,

S – стартовый символ грамматики, являющийся нетерминалом.

В контекстно-свободной грамматике правила имеют вид:

, где , ,  - словарь грамматики .

Перечень правил, описывавших контекстно-свободную грамматику языка KAV-2020, представлен в таблице 4.1.

Таблица 4.2 – Перечень правил грамматики языка KAV-2020

|  |  |  |
| --- | --- | --- |
| Нетерминальный символ | Цепочки правил | Какие правила порождает |
| S-> | sfi(F){N}S  nfi(F){N}S  sfi(){N}S  nfi(){N}S  m{N} | Правила, описывающие общую структуру программы |
| F-> | si,F  si  ni,F  ni | Правила, описывающие наличие списка параметров функции |
| N-> | tni;N  tCi;N  tsi;N  i=E;N  c(i,i);N  c(l,l);N  b;b;N  oE;N  ri;  rl;  tni;  tsi;  i=E;  b;  oE;  I(iLi){oE;}G{oE;}N  I(iLi){oE;}G{oE;}  T(i,i);N | Правила, описывающие возможные конструкции |
| E-> | i  iE  l  iM  lM  d  dE  a  aE  (W)  (W)M  (E)M  i() | Правила, описывающие выражения |

Продолжение таблицы 4.2

|  |  |  |
| --- | --- | --- |
| Нетерминальный символ | Цепочки правил | Какие правила порождает |
|  | i()M  p(W)  p(W)M  k(W)  k(W)M |  |
| W-> | i  l  i,W  l,W | Правила, описывающие принимаемые параметры функции |
| M-> | +E  -E  \*E  /E | Правила, описывающие арифметические знаки |
| Y-> | !  >  <  &  $  ? | Правила правила описывающие логические знаки |

**4.3 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.

Схема работы автомата с магазинной памятью представлена на рисунке 4.2.

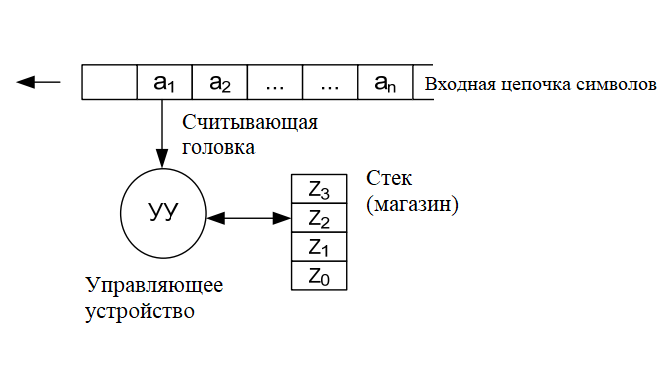
****

Рисунок 4.2 Схема автомата с магазинной памятью

**4.4 Основные структуры данных**

Основные структуры данных синтаксического анализатора включают в себя структуру магазинного автомата и структуру грамматики Грейбах, описывающая правила языка KAV-2020. Структура синтаксического анализатора представлена в приложении Г.

**4.5 Описание алгоритма синтаксического разбора**

Алгоритм синтаксического разбора:

* происходит поиск и выделение синтаксических конструкций в исходном тексте (разбор);
* распознавание (проверка правильности) синтаксических конструкций;
* выявление ошибок и продолжение процесса распознавания после обработки ошибок;
* в случае отсутствия ошибок, формируется дерево разбора.

**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 |
| 50 | Неверная структура программы |
| 51 | Ошибка в параметрах функции |
| 52 | Ошибочный оператор |
| 53 | Ошибка в выражении |
| 54 | Ошибка в параметрах вызываемой функции |
| 55 | Ошибка в построении выражения |
| 56 | Неправильное использование знаков арифметических операций или операции присваивания |

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

Входными данными для синтаксического анализатора являются таблицы лексем и идентификаторов. Также используется описание грамматики в форме Грейбах. Результаты работы лексического разбора, а именно дерево разбора и протокол работы автомата с магазинной памятью выводятся в журнал работы программы.

**4.8 Принцип обработки ошибок**

Принцип заключается в том, что синтаксический анализатор перебирает все возможные правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем. При возникновении ошибки синтаксический анализатор откатиться назад до правила, при помощи которого разбор был успешным, если это возможно. После чего применяются последующие правила из грамматики. В случае, если правило невозможно подобрать, выводится одно из сообщений об ошибке, представленное в таблице 4.2. Все ошибки записываются в общую структуру ошибок, а также отображаются на консоли. Если в процессе анализа находятся более трёх ошибок, то анализ останавливается.

**4.9 Контрольный пример**

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

**5. Разработка семантического анализатора**

**5.1 Структура семантического анализатора**

Семантический анализатор в трансляторе языка KAV-2020 не выделен в отдельную фазу. Проверка на ошибки в исходном коде производится на этапе лексического анализа.

**5.2 Функции семантического анализатора**

Основные действия семантического анализатора:

* проверка соблюдения в исходной программе семантических правил входного языка;
* дополнение внутреннего представления программы в компиляторе операторами и действиями, неявно предусмотренными семантикой входного языка;
* проверка элементарных семантических (смысловых) норм языка программирования.

Семантические правилаязыка KAV-2020 представлены в главе 1 пункте 1.16.

**5.3 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 |
| 80 | Превышено максимальное количество параметров при вызове функции |
| 81 | Попытка переопределения идентификатора |
| 82 | Отсутствует объявление идентификатора |
| 83 | Отсутствует точка входа main |
| 84 | Точка входа main должна быть единственной |
| 85 | Несоответствие типов |
| 86 | Запрещено присваивать значение функции |
| 87 | Запрещено делить на ноль |
| 88 | Использование стандартных функций без подключения стандартной библиотеки |
| 89 | Попытка подключить стандартную библиотеку более одного раза |
| 90 | Попытка проведения операций со строковым типом данных |
| 91 | Тип возвращаемого значения и тип функции не совпадают |

Продолжение таблицы 5.1

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 92 | Неправильное количество параметров при вызове функции |
| 93 | Неправильный тип параметров при вызове функции |
| 94 | Превышено допустимое количество параметров функции |
| 95 | Неправильный вызов функции |

**5.4 Принцип обработки ошибок**

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

**5.5 Контрольный пример**

Контрольный пример заключается в тестировании функций семантического анализатора при наличии соответствующих ошибок в исходном коде. Тестирование представлено в таблице 5.3.

Таблица 5.3 – Примеры диагностики ошибок

|  |  |
| --- | --- |
| Исходный код с ошибкой | Сообщение об ошибке |
| main  {  x = 2;  let int x;  print(x);  } |  |
| main  {  x = 2;  main  {  let int x;  }  } |  |
| main  {  let int k;  k = 5/0;  print(k);  } |  |

# **6. Вычисление выражений**

**6.1. Выражения, допускаемые языком**

В языке программирования KAV-2020 допускается вычисление выражений с целочисленными типами данных. В выражениях поддерживаются арифметические операции, такие как +, -, \*, / и (), логические операции, такие как <, >, !, &, {, }, а также вызовы функций из них. В выражениях могут присутствовать операции, которые описаны в пункте 1.12.

Каждая операция в выражении имеет свой приоритет. Приоритетность арифметических операций представлена в таблице 6.1.

Таблица 6.1 – Приоритетность операций

|  |  |  |
| --- | --- | --- |
| Операция | Приоритет | Порядок выполнения |
| () | 1 | Наивысший приоритет (выполняется в первую очередь) |
| \* | 2 | Вторая степень приоритетности |
| / | 2 | Вторая степень приоритетности |
| + | 3 | Наименьший приоритет (выполняется в последнюю очередь) |
| - | 3 | Наименьший приоритет (выполняется в последнюю очередь) |

**6.2. Польская запись и принцип ее построения**

Все выражения языка KAV-2020 преобразовываются к обратной польской записи.

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 преобразования выражений к 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.

## **6.3. Программная реализация обработки выражений**

Программная реализация преобразования выражений в обратный польский формат основана на структуре, представленной на рисунке 6.1. Изначально происходит поиск выражения в таблице лексем функцией search. Преобразование выражений в польскую запись реализовано с помощью функции CreatePolishNotation, результатом работы которой является промежуточный код, который записывается в общий протокол работы транслятора.

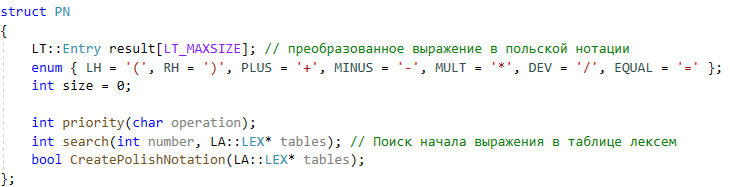


Рисунок 6.1 — Фрагмент кода для преобразования выражений

## **6.4. 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 |
| 1\*(2-3)+4/5 |  |  |
| \*(2-3)+4/5 |  | 1 |
| (2-3)+4/5 | \* | 1 |
| 2-3)+/5 | \*( | 1 |
| -3)+4/5 | \*( | 12 |

Продолжение таблицы 6.2

|  |  |  |
| --- | --- | --- |
| Выражение | Стек | Результат |
| 3)+4/5 | \*(- | 12 |
| )+4/5 | \*(- | 123 |
| +4/5 | \* | 123- |
| 4/5 | **+** | 123-\* |
| /5 | **+** | 123-\*4 |
| 5 | **+/** | 123-\*4 |
|  | **+/** | 123-\*45 |
|  |  | 123-\*45/+ |

# **7. Генерация кода**

## **7.1. Структура генератора кода**

В языке KAV-2020 генерация кода является заключительным этапом трансляции. Генератор принимает на вход таблицы лексем и идентификаторов, полученные в результате лексического анализа. В соответствии с таблицей лексем строится выходной файл на языке C#, который будет являться результатом работы транслятора. В случае возникновения ошибок генерация кода не будет осуществляться. Структура генератора кода KAV-2020 представлена на рисунке 7.1.

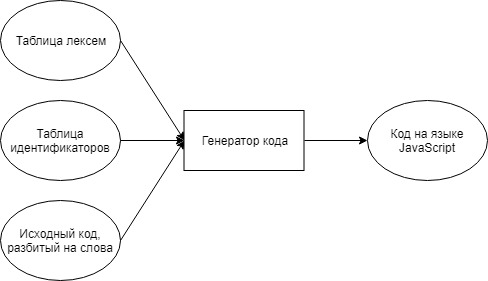


Рисунок 7.1 Структура генератора кода

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

Язык KAV-2020 требует указывать тип данных при объявлении идентификатора с использованием ключевого слова init. Язык C# так же требует указывать тип данных при объявлении. Соответствия между типами данных идентификаторов языка KAV-2020 и языка C# представлены в таблице 7.1.

Таблица 7.1 – Соответствие типов идентификаторов языка KAV-2020 и языка C#

|  |  |
| --- | --- |
| Тип идентификаторов языка KAV-2020 | Тип данных языка C# |
| int (беззнаковый целый) | uint (беззнаковый целый) |
| char (символьный) | char (символьный) |
| string (строковый) | string (строковый) |

## **7.3. Статическая библиотека**

Для языка KAV-2020 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 KAV-2020:

* 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;

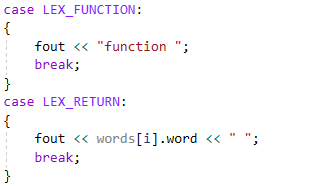


Рисунок 7.2 — Часть кода генерации

## **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.

# **8. Тестирование транслятора**

## **8.1. Общие положения**

В результате обработки исходного кода программы, представленного в приложении А, транслятор языка KAV-2020 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 KAV-2020 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 |
| main  {  let int m;  m = root(4);  print(m);  } |  |
| int def fu(int s, int)  {  let int l;  l = s;  return d;  }  main  {  let int l;  l = func(6);  } |  |
| main  {  let int 888p;  888p = 36;  print(888);} |  |

# **Заключение**

Основной целью курсовой работы было понять принцип работы языков программирования. Данный курсовой проект позволил совместить закрепление знаний сразу по двум языкам программирования, таких как C++ и JavaScript. Были усвоены такие понятия как синтаксический, лексический и семантический анализаторы и многие другие.

В итоге был получен примитивный язык программирования KAV-2020, который не имеет сложных конструкций, которые реализованы на сегодняшний день во многих других языках программирования.

Окончательная версия языка KAV-2020 включает:

* 2 типа данных;
* Поддержка операции вывода;
* Возможность вызова функций стандартных библиотек;
* Наличие 2 арифметических операторов для вычисления выражений;
* Наличие условного оператора
* Наличие двоичного и восьмеричного представления числа
* Наличие операций сравнения
* Структурированная система для обработки ошибок пользователя.

Основные характеристики транслятора KAV-2020:

* Возможность обработки 32 ошибок;
* Реализация 28 конечных автоматов;
* Реализация 58 цепочек правил грамматики;
* Наличие порядка 2600 строк кода;

**Список использованных источников**

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

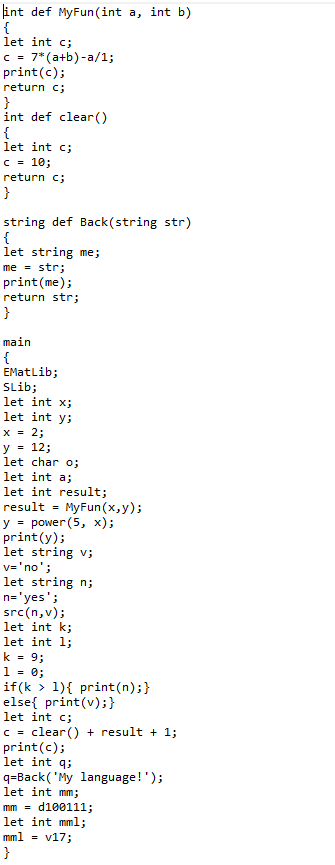
2 Ахо А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003. – 768с.

3 Никлаус Вирт Построение компиляторов / Пер. с англ. Борисов Е. В., Чернышов Л. Н. – М.: ДМК Пресс, 2010. – 192 с.: с ил.

4 Польская запись [Электронный ресурс] – Режим доступа: https://studfiles.net/preview/2792990/

5 Классификация языков программирования [Электронный ресурс] – Режим доступа: http://bourabai.kz/alg/classification.htm

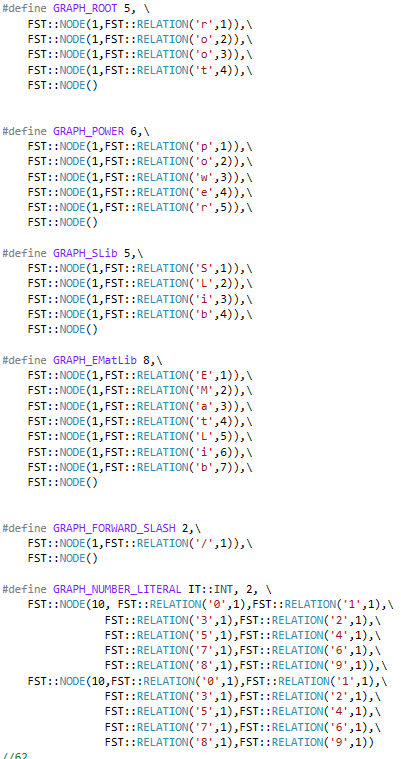
**Приложения A**

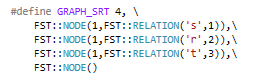
****

**Приложение Б**

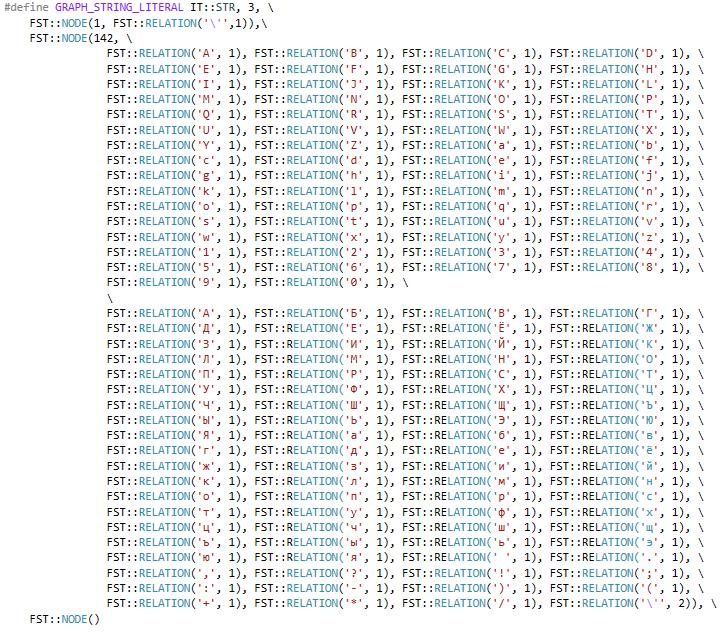
****

Продолжение приложения Б

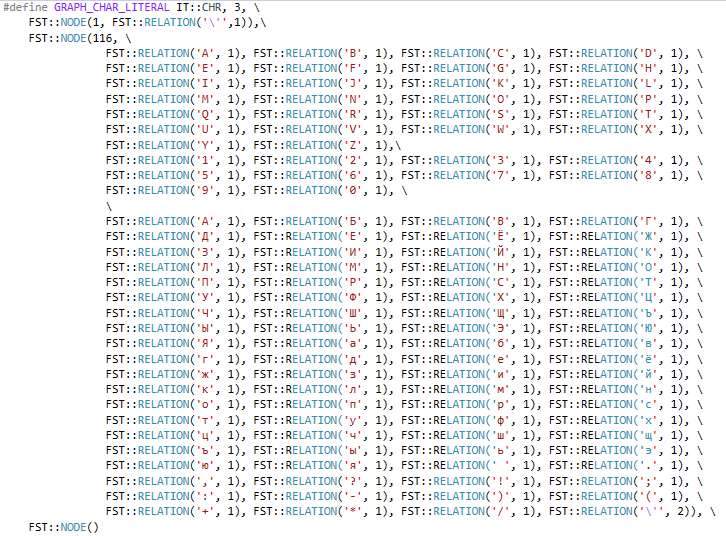
****

****

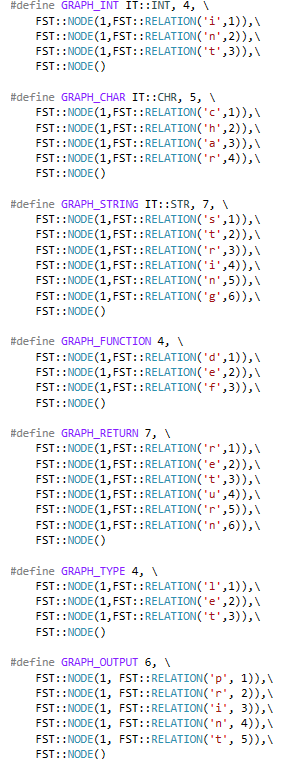
Продолжение приложения Б

****

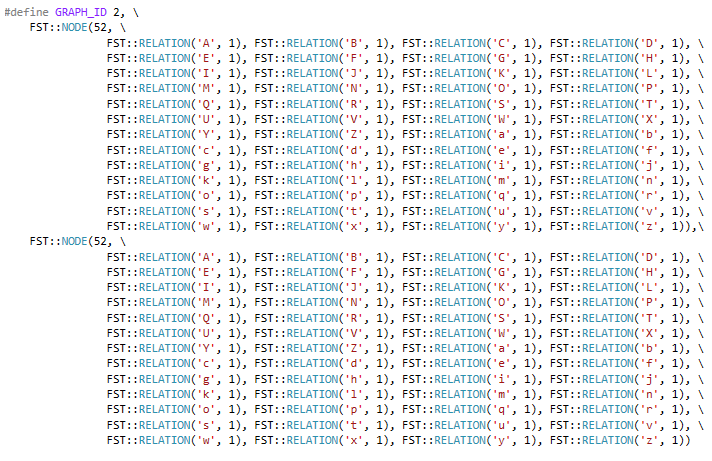
Продолжение приложения Б

****

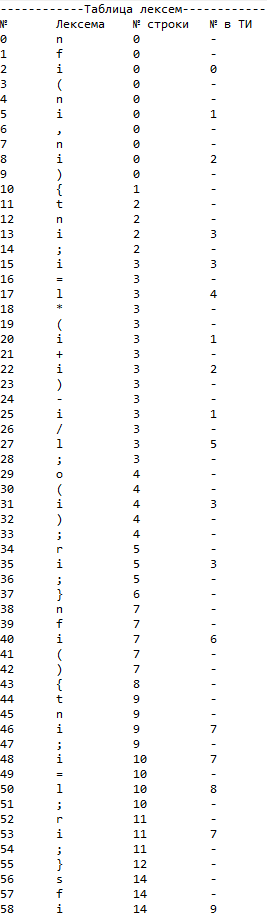
Продолжение приложения Б

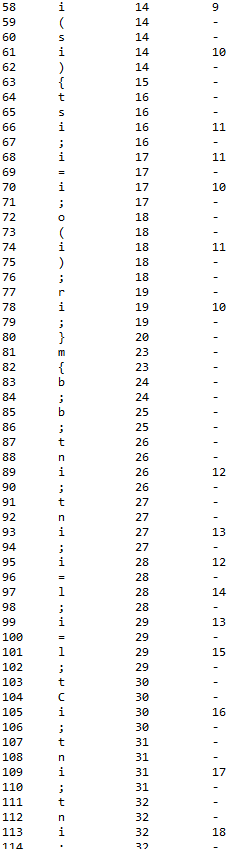
****

Продолжение приложения Б

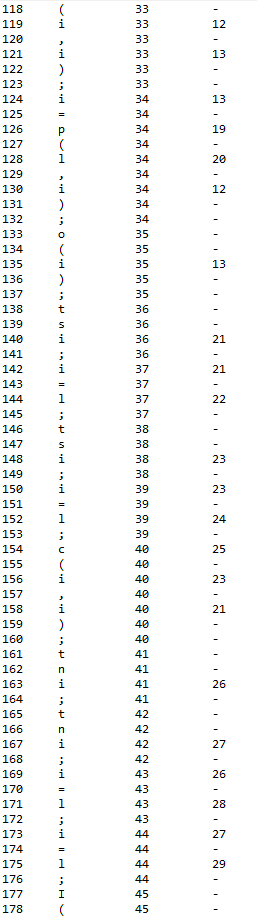
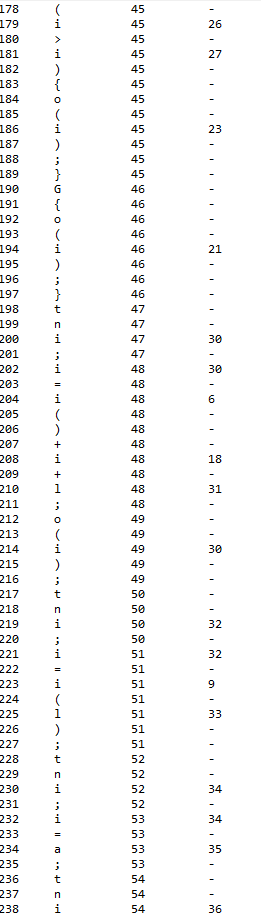
****

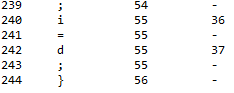
# **Приложение В**

****

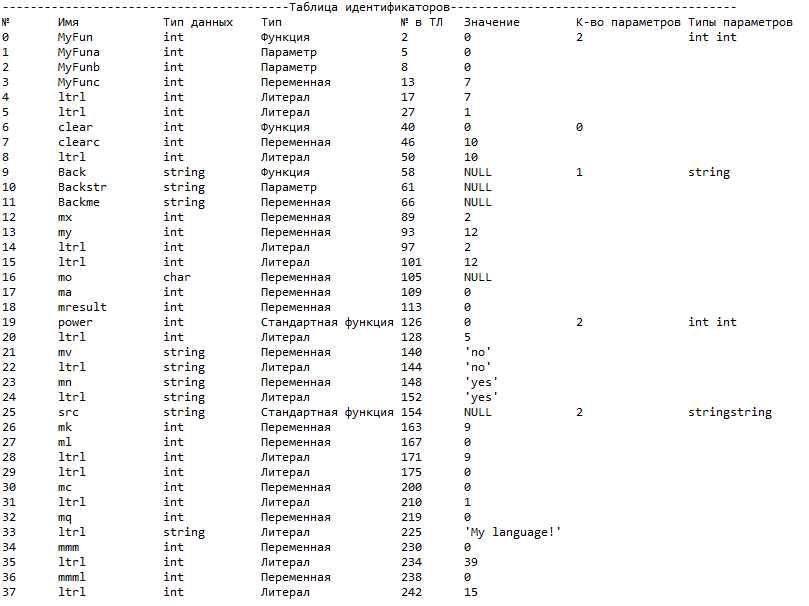
****

Продолжение приложение В

****

****

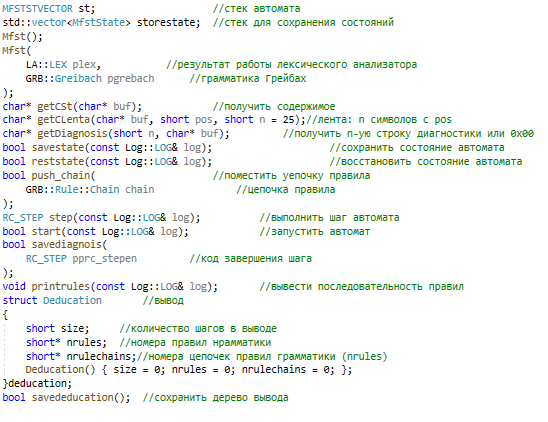
Продолжение приложение В

****

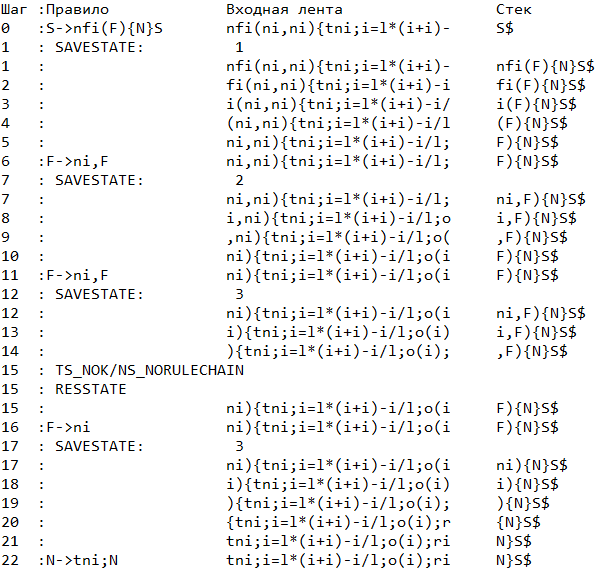
**Приложение Г**

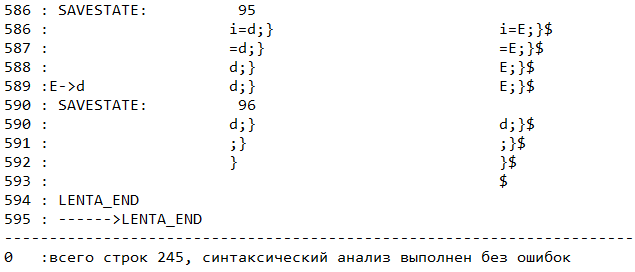
****

Продолжение приложения Г

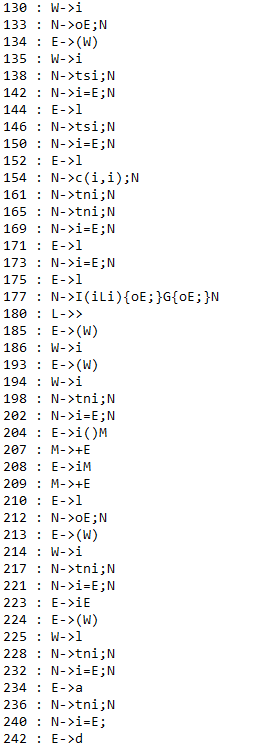
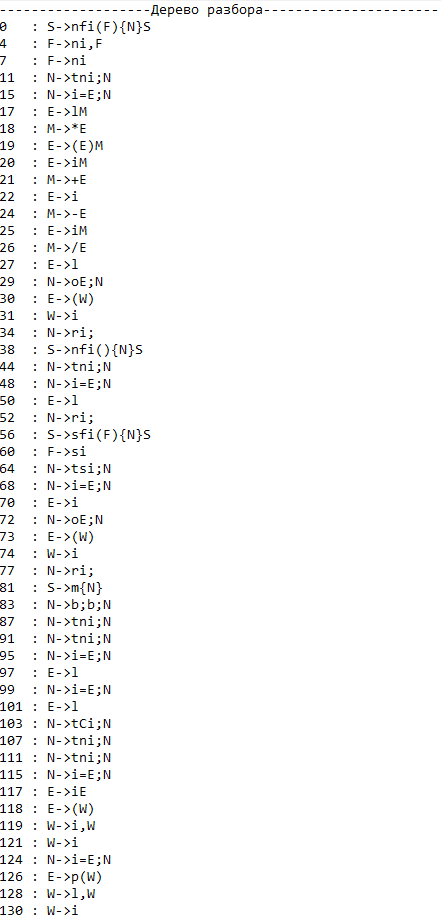
****

**Приложение Д**

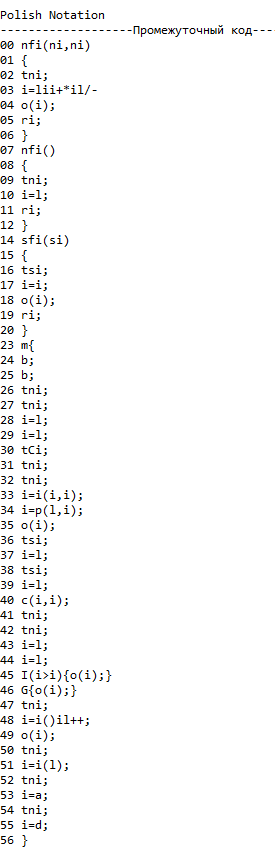
****

****

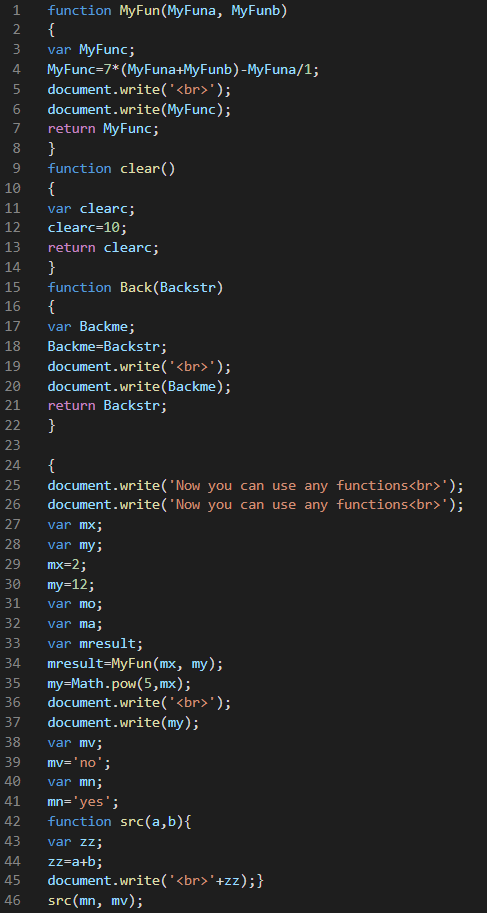
Продолжение приложения Д

****

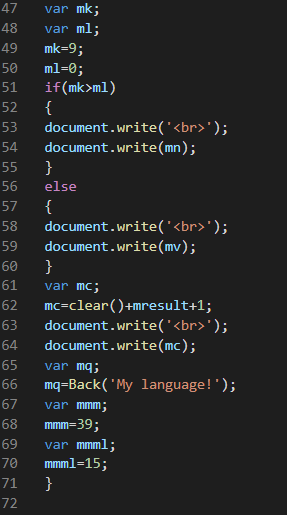
**Приложение Е**

****

# **Приложение Ж**

****

Продолжение приложения Ж

****