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 AEV-2020»

Выполнил студент Астапкина Екатерина 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

**Оглавление**

[Введение 4](#_Toc58737272)

[1. Спецификация языка программирования 5](#_Toc58737273)

[1.1 Характеристика языка программирования 5](#_Toc58737274)

[1.2 Определение алфавита языка программирования 5](#_Toc58737275)

[1.3 Применяемые сепараторы 5](#_Toc58737276)

[1.4 Применяемые кодировки 5](#_Toc58737277)

[1.5 Типы данных 6](#_Toc58737278)

[1.6 Преобразование типов данных 6](#_Toc58737279)

[1.7 Идентификаторы 7](#_Toc58737280)

[1.8 Литералы 7](#_Toc58737281)

[1.9 Объявление данных 7](#_Toc58737282)

[1.10 Инициализация данных 8](#_Toc58737283)

[1.11 Инструкции языка 8](#_Toc58737284)

[1.12 Операция языка 9](#_Toc58737285)

[1.13. Выражения и их вычисление 9](#_Toc58737286)

[1.14. Конструкции языка 9](#_Toc58737287)

[1.15 Область видимости идентификаторов 10](#_Toc58737288)

[1.16 Семантические проверки 10](#_Toc58737289)

[1.17 Распределение оперативной памяти на этапе выполнения 11](#_Toc58737290)

[1.18 Стандартная библиотека и ее состав 11](#_Toc58737291)

[1.19 Ввод и вывод данных 11](#_Toc58737292)

[1.20 Точка входа 11](#_Toc58737293)

[1.21 Препроцессор 11](#_Toc58737294)

[1.22 Соглашения о вызовах 11](#_Toc58737295)

[1.23 Объектный код 12](#_Toc58737296)

[1.24 Классификация сообщений транслятора 12](#_Toc58737297)

[1.25 Контрольный пример 12](#_Toc58737298)

[2. Структура транслятора 13](#_Toc58737299)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 13](#_Toc58737300)

[2.2 Перечень входных параметров транслятора 14](#_Toc58737301)

[2.3 Протоколы, формируемые транслятором 14](#_Toc58737302)

[3. Разработка лексического анализатора 15](#_Toc58737303)

[3.1 Структура лексического анализатора 15](#_Toc58737304)

[3.2 Контроль входных символов 15](#_Toc58737305)

[3.3 Удаление избыточных символов 16](#_Toc58737306)

[3.4 Перечень ключевых слов 16](#_Toc58737307)

[3.5. Основные структуры данных 17](#_Toc58737308)

[3.6 Структура и перечень сообщений лексического анализатора 18](#_Toc58737309)

[3.7 Принцип обработки ошибок 19](#_Toc58737310)

[3.8 Параметры лексического анализатора 19](#_Toc58737311)

[3.9 Алгоритм лексического анализа 19](#_Toc58737312)

[3.10 Контрольный пример 20](#_Toc58737313)

[4. Разработка синтаксического анализатора 21](#_Toc58737314)

[4.1 Структура синтаксического анализатора 21](#_Toc58737315)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 21](#_Toc58737316)

[4.3 Построение конечного магазинного автомата 23](#_Toc58737317)

[4.4 Основные структуры данных 25](#_Toc58737318)

[4.5 Описание алгоритма синтаксического разбора 25](#_Toc58737319)

[4.6 Структура и перечень сообщений синтаксического анализатора 25](#_Toc58737320)

[4.7 Параметры синтаксического анализатора и режимы его работы 25](#_Toc58737321)

[4.8 Принцип обработки ошибок 26](#_Toc58737322)

[4.9 Контрольный пример 26](#_Toc58737323)

[5. Разработка семантического анализатора 27](#_Toc58737324)

[5.1 Структура семантического анализатора 27](#_Toc58737325)

[5.2 Функции семантического анализатора 27](#_Toc58737326)

[5.3 Структура и перечень сообщений семантического анализатора 27](#_Toc58737327)

[5.4 Принцип обработки ошибок 28](#_Toc58737328)

[5.5 Контрольный пример 28](#_Toc58737329)

[6. Вычисление выражений 30](#_Toc58737330)

[6.1 Выражения, допускаемые языком 30](#_Toc58737331)

[6.2 Польская запись и принцип ее построения 30](#_Toc58737332)

[6.3 Программная реализация обработки выражений 31](#_Toc58737333)

[6.4 Контрольный пример 31](#_Toc58737334)

[7. Генерация кода 32](#_Toc58737335)

[7.1 Структура генератора кода 32](#_Toc58737336)

[7.2 Представление типов данных в оперативной памяти 32](#_Toc58737337)

[7.3 Статическая библиотека 33](#_Toc58737338)

[7.4 Особенности алгоритма генерации кода 33](#_Toc58737339)

[7.5 Входные параметры генератора кода 33](#_Toc58737340)

[7.6 Контрольный пример 33](#_Toc58737341)

[8. Тестирование транслятора 34](#_Toc58737342)

[8.1 Общие положения 34](#_Toc58737343)

[8.2 Результаты тестирования 34](#_Toc58737344)

[Заключение 36](#_Toc58737345)

[Графический материал 37](#_Toc58737346)

[Список использованных источников 38](#_Toc58737347)

[Приложение А 39](#_Toc58737348)

[Приложение Б 40](#_Toc58737349)

[Приложение В 44](#_Toc58737350)

[Приложение Г 46](#_Toc58737351)

[Приложение Д 48](#_Toc58737352)

[Приложение Е 51](#_Toc58737353)

[Приложение Ж 52](#_Toc58737354)

# **Введение**

Основной целью данного курсового проекта является разработка транслятора для языка программирования AEV-2020. Главная задача транслятора заключается в том, чтобы сделать программу, написанную на языке программирования AEV-2020, понятной компьютеру. 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 AEV-2020 предназначен для работы с консолью, выполнения простейших арифметический действий, операций над логическими переменными.

# **Спецификация языка программирования**

## **1.1 Характеристика языка программирования**

Язык программирования AEV-2020 – это универсальный язык высокого уровня. Он предназначен для выполнения простейших арифметических действий. Язык является процедурным, компилируемым, не объектно-ориентированным, строго типизируемым.

## **1.2 Определение алфавита языка программирования**

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

## **1.3 Применяемые сепараторы**

Сепараторы нужны для разделения операций языка. Символы, которые являются сепараторами в языке программирования AEV-2020 представлены в таблице 1.1.

Таблица 1.1 – Символы сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение |
| ; | Разделитель программных конструкций |
| = | Оператор присваивания |
| ‘ ‘ (пробел) | Разделитель цепочек. Допускается везде, кроме названий идентификаторов и ключевых слов |
| , | Разделитель параметров функций |
| >  <  &  !  ]  [ | 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 AEV-2020 используется кодировка Windows-1251, представленная на рисунке 1.1.

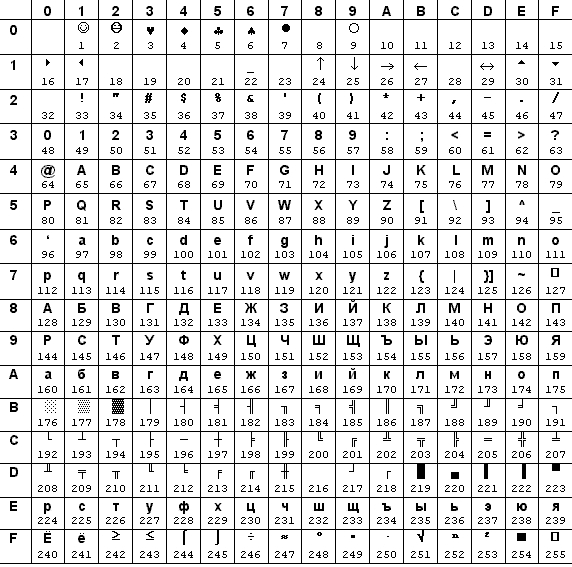


Рисунок 1.1 Кодировка Windows 1251

**1.5 Типы данных**

В языке программирования AEV-2020 используется два типа данных: беззнаковый целочисленный и символьный. Пользовательские типы данных не поддерживаются. Описание типов данных, предусмотренных в данном языке представлено в таблице 1.2.

Таблица 1.2 – Типы данный языка программирования AEV-2020

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| num | Фундаментальный тип данных. Предусмотрен для объявления целочисленных беззнаковых данных. Автоматически инициализируется нулевым значением. |
| char | Фундаментальный тип данных. Предусмотрен для объявления символов. Автоматическая инициализация нулевым значением. |

**1.6 Преобразование типов данных**

В языке программирования AEV-2020 преобразование типов данных не поддерживается, все типы данных определены однозначно. Попытка преобразования типов данных приведет к семантической ошибке.

## **1.7 Идентификаторы**

Идентификаторы используются для именования функций, параметров и переменных. В языке программирования AEV-2020 не предусмотрены зарезервированные идентификаторы. В имени идентификатора допускаются символы латинского алфавита верхнего и нижнего регистра, максимальная длина имени - 10 символов. В случае превышения максимального значения длина имени идентификатора усекается до 10 символов. Идентификаторы не должны совпадать с ключевыми словами.

## **1.8 Литералы**

Литерал – лексема, которая непосредственно представляет некоторое значение. В языке программирования AEV-2020 существует два типа литералов: целые и символьные. Правило записи литералов представлено в таблице 1.3.

Таблица 1.3 – Правило записи литералов

|  |  |
| --- | --- |
| Тип литерала | Описание |
| Числовые | Целочисленные неотрицательные литералы в десятичной системе счисления, инициализируются 0. Не имеют дробных частей, только положительные. |
| Символьные | Символы, заключённые в ' '(одинарные кавычки), инициализируются пустой строкой, может содержать только 1 символ |

Ограничения на строковые литералы языка AEV-2020: внутри литерала не допускается использование одинарных и двойных кавычек. Ограничения на целочисленные литералы: не могут начинаться с нуля, если их значение не ноль.

## **1.9 Объявление данных**

Областью видимости переменной в языке программирования AEV-2020 является блок функции (область видимости «сверху вниз», по принципу С++), в которой она определена. В языке программирования AEV-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:

init <тип\_даных> <идентификатор> = [<литерал> | <идентификатор> | <выражение>];

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

## **1.10 Инициализация данных**

В языке программирования AEV-2020 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 в языке программирования AEV-2020 представлены в таблице 1.4.

Таблица 1.4 – Способы инициализации переменных

|  |  |
| --- | --- |
| Вид инициализации | Примечание |
| init <тип\_данных> <идентификатор>; | Автоматическая инициализация: переменные типа num инициализируются нулём, переменные типа char – пустым символом |
| <идентификатор> = <значение>; | Присваивание переменной значения |

## **1.11 Инструкции языка**

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

Таблица 1.5 – Инструкции языка

|  |  |
| --- | --- |
| Инструкция | Синтаксис |
| Объявление переменной | init <тип данных> <идентификатор>; |
| Присваивание | <идентификатор>=<значение>|<идентификатор>|  <выражение>; |
| Главная функция | main  {  <инструкции языка>  } |
| Вызов функции | <идентификатор функции>(<идентификатор>|  <литерал, …>); |
| Возврат функции | return <идентификатор>|<литерал>; |
| Объявление функции | <тип данных> function <идентификатор>  {  <инструкции языка>  } |
| Вывод данных | out (<идентификатор> | <литерал> | <выражение>); |
| Оператор цикла | until(<условие>)  {…} |

## **1.12 Операция языка**

Операции сравнения в языке программирования AEV-2020 применимы к беззнаковым целым и символьным 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 AEV-2020 может выполнять операции, представленные в таблице 1.6.

Таблица 1.6 – операции языка программирования AEV-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. Результатом вычисления выражения является величина определённого типа. В языке программирования выражения должны быть записаны в одну строку. Круглые скобки в выражении используются для изменения приоритета операций. Не допускается запись двух подряд идущих арифметических операций.

## **1.14. Конструкции языка**

Программа на языке AEV-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 AEV-2020 представлены в таблице 1.7.

Таблица 1.7 – Программные конструкции языка

|  |  |
| --- | --- |
| Название | Синтаксис |
| Главная функция (точка входа в приложение) | main  {…} |
| Функция | <тип данных> function<идентификатор>(<тип данных> <идентификатор>, …)  {  …  return <выражение>;  } |
| Оператор цикла | until (<условие>)  { … } |

## **1.15 Область видимости идентификаторов**

В языке программирования AEV-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 программирования AEV-2020, 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 | Проводить арифметические операции со строковым типом данных запрещено |
| 5 | Тип данных возвращаемой функции должен совпадать с типом при ее объявлении |
| 6 | Не должно быть объявлений идентификаторов с одинаковыми именами в одном и том же блоке кода |
| 7 | Деление на ноль запрещено |
| 8 | Наличие только одной точки входа main |
| 9 | Правильность передаваемых в функцию параметров: количество, типы |
| 10 | При использовании стандартных функций должна быть подключена стандартная библиотека и только один раз |
| 11 | Присваивать значение функции запрещено |

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

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**

В языке программирования AEV-2020 присутствует стандартная библиотека, которая подключается автоматически при трансляции исходного кода в язык JavaScript. Функции стандартной библиотеки с описанием представлены в таблице 1.9.

Таблица 1.9 – Cостав стандартной библиотеки

|  |  |  |  |
| --- | --- | --- | --- |
| Имя функции | Возвращаемое значение | Принимаемые параметры | Описание |
| pow | num | num a – число,  num b – число | Возводит число a в степень b |
| avr | num | num x – число,  num y – число,  num z – число | Возвращает среднее значение трех чисел |

## **1.19 Ввод и вывод данных**

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

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

## **1.20 Точка входа**

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

**1.21 Препроцессор**

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

**1.22 Соглашения о вызовах**

Соглашение о вызовах — это протокол для передачи аргументов функциям. В языке программирования AEV-2020 используется соглашение о вызовах stdcall. Особенности stdcall:

* все параметры функции передаются через стек;
* память высвобождает вызываемый код;
* занесение в стек параметров идёт справа налево.

## **1.23 Объектный код**

Исходный код языка программирования AEV-2020 транслируется в язык JavaScript, а после интерпретируется браузером.

**1.24 Классификация сообщений транслятора**

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

Таблица 1.10 – Классификация сообщений транслятора

|  |  |
| --- | --- |
| Интервал | Описание ошибок |
| 0 – 9 | Системные ошибки |
| 10 – 49 | Ошибки входных параметров |
| 50 – 89 | Ошибки открытия и чтения файлов |
| 90 – 99 | Ошибки лексического анализа |
| 100 – 299 | Ошибки синтаксического анализа |
| 300 – 399 | Ошибки семантического анализа |

**1.25 Контрольный пример**

Контрольный пример показывает результат работы всех этапов трансляции исходного кода на языке программирования AEV-2020 в язык JavaScript.

Контрольный пример представлен в Приложении А.

* + - 1. **Структура транслятора**

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

Транслятор языка программирования AEV- 2020 преобразует исходный код программы в код целевого языка – JavaScript. Процесс трансляции разделён на отдельные фазы, которые взаимодействуют между собой и выполняют отведённые им функции. На всех фазах трансляции используются таблица лексем и таблица идентификаторов. 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 ассемблерного кода на основе полученных данных на предыдущих этапах трансляции. На вход генератора подаются таблица лексем и таблица идентификаторов, на основе которых генерируется файл с ассемблерным кодом.

## **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 |
| -log: | Указывает транслятору в какой файл выводить протокол работы транслятора. |

**2.3 Протоколы, формируемые транслятором**

В ходе работы программы формируются протоколы работы лексического, синтаксического и семантического анализаторов, которые содержат в себе перечень протоколов работы. Перечень протоколов, формируемых транслятором, представлен в таблице 2.2.

Таблица 2.2 – Протоколы, формируемые транслятором

|  |  |
| --- | --- |
| Протокол | Описание протокола |
| Файл журнала, заданный параметром “-log:” | Результат работы транслятора языка программирования AEV-2020. Содержит таблицы лексем и идентификаторов, протокол работы синтаксического анализатора, дерево разбора, а также результат работы алгоритма преобразования выражений к польской записи |

* + - 1. **Разработка лексического анализатора**

**3.1 Структура лексического анализатора**

Первая фаза трансляции называется лексическим анализом, а программа, её реализующая, – лексическим анализатором (сканером). На вход лексического анализатора подаётся исходный код входного языка. Лексический анализатор выделяет в этой последовательности простейшие конструкции языка. Примеры лексических единиц: идентификаторы, числа, символы операций, служебные слова и т.д. Лексический анализатор преобразует исходный текст, заменяя лексические единицы их внутренним представлением – лексемами. В результате работы лексического анализатора формируется таблица идентификаторов и таблица лексем, модифицируется протокол работы транслятора.

Структура лексического анализатора представлена на рисунке 3.1.

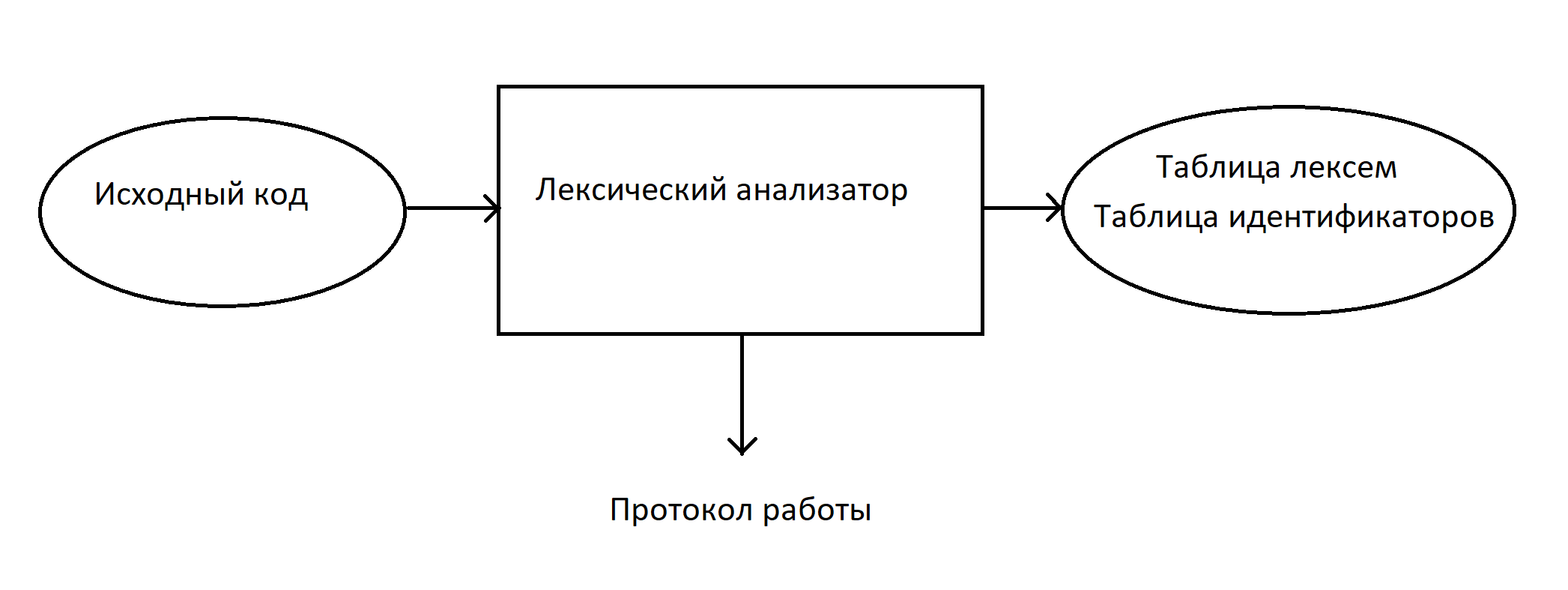


Рисунок 3.1 Структура лексического анализатора

**3.2 Контроль входных символов**

Для удобной работы с исходным кодом, при передаче его в лексический анализатор, все символы разделяются по категориям. Таблица для контроля входных символов языка AEV-2020 представлена на рисунке 3.2.

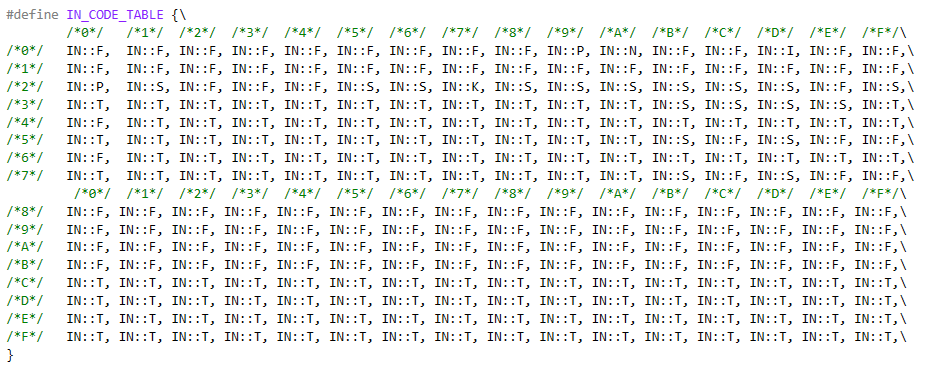


Рисунок 3.2 Таблица для контроля входных символов

Каждому элементу соответствует значение в шестнадцатеричной системе счисления - такое же, как и в таблице Windows-1251 (изображено на рисунке 1.1).

Каждый считанный из файла с исходным кодом символ проверяется на допустимость. Если символ в таблице обозначен как T, то это разрешенный символ, который будет записан в строку, если символ обозначен буквой F, то его позиция в исходном коде записывается в структуру ошибок. В случае обнаружения недопустимого символа будет сгенерирована ошибка и она будет выведена в консоль вместе с позицией в исходном коде. Символ, обозначенный буквой I – игнорируемый, S – символы-сепараторы, N – новая строка, P – пробел, K – одинарная кавычка (‘).

**3.3 Удаление избыточных символов**

Избыточными в языке программирования AEV-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. Посимвольно считывается файл с исходным кодом программы;
2. В конце каждой итерации посимвольного считывания текущий символ сохраняется в буфер;
3. встреча пробела включает проверку предыдущего 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**

Лексический анализатор преобразует исходный текст, заменяя лексические единицы лексемами для создания промежуточного представления исходной программы. Соответствие токенов и лексем приведено в таблице 3.1.

Таблица 3.1 – Таблица соответствия токенов и лексем

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Примечание |
| Сепараторы | S | Символы ; , {} () +-\*/ = >< [] ! & |
| num | n | Беззнаковый целый тип данных |
| char | c | Символьный тип данных |
| init | t | Объявление переменной |
| Идентификатор | i | Идентификатор, длина которого 10 символов |
| Литерал | l | Литерал любого доступного типа |
| function | f | Объявление функции |
| return | r | Выход из функции |
| out | o | Оператор вывода |

Продолжение таблицы 3.1

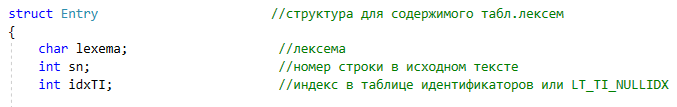
|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Примечание |
| main | m | Главная функция |
| until | u | Оператор цикла |
| ; | ; | Разделение выражений |
| , | , | Разделение параметров функций |
| { | { | Начало блока/тела функции |
| } | } | Конец блока/тела функции |
| ( | ( | Передача параметров в функцию, приоритет операций |
| ) | ) | Закрытие блока для передачи параметров, приоритет операций |
| + | + | Знаки арифметических операций |
| - | - |
| \* | \* |
| / | / |
| = | = | Оператор присваивания |
| > | > | Знаки логических операторов |
| < | < |
| ] | ] |
| [ | [ |
| & | & |
| ! | ! |

Для каждого токена соответствует автомат, по которому происходит разбор выражения. Автомат является детерменированным, то есть имеет конечное количество состояний. Перечень цепочек, соответствующих им автоматов и лексем, представлен в приложении Б.

**3.5. Основные структуры данных**

Основными структурами данных лексического анализатора языка программирования AEV-2020 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, количество параметров (если функция).

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



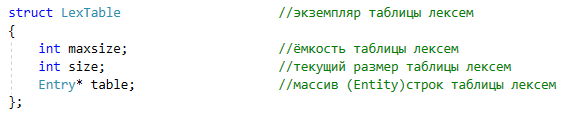
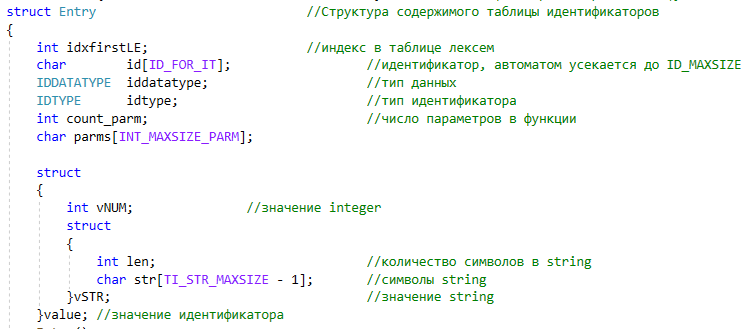


Рисунок 3.3 Структуры таблиц лексем AEV-2020

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



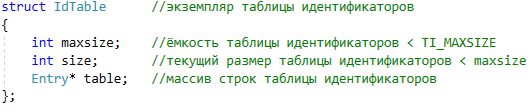


Рисунок 3.4 Структуры таблиц идентификаторов AEV-2020

**3.6 Структура и перечень сообщений лексического анализатора**

В случае нахождения лексическим анализатором ошибки, происходит запись в протокол номера ошибки и диагностического сообщения. Перечень сообщений, формируемых лексическим анализатором в ходе своей работы, представлен в таблице 3.2.

Таблица 3.2 — Перечень сообщений лексического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 90 | Превышен размер таблицы идентификаторов |
| 91 | Превышен размер таблицы лексем |
| 92 | Слово не разобрано |
| 94 | В символьной переменной не может быть больше одного символа |

**3.7 Принцип обработки ошибок**

Для обработки ошибок лексический анализатор использует таблицу с сообщениями. Если в процессе лексического анализа обнаруживается ошибка, то заполняется структура, содержащая ошибку, произошедшую в процессе лексического анализа, номер строки в исходном коде. Максимально допустимое количество ошибок в лексическом анализе ограничено числом 1. В случае обнаружения ошибки в консоль будет выведена информация об данной ошибке и дальнейший процесс будет прерван.

**3.8 Параметры лексического анализатора**

Результаты работы лексического анализатора, а именно таблицы лексем и идентификаторов выводятся как в файл журнала, так и в командную строку. Описание входного параметра для выводов результатов приведено в таблице 2.1.

**3.9 Алгоритм лексического анализа**

Лексический анализ является первой и наиболее простой фазой трансляции. Алгоритм лексического анализатора заключается в следующем:

* сначала проверяется входной поток символов программы на исходном языке на допустимость, удаляются лишние пробелы;
* 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 выводится сообщение об ошибке с указанием номера строки;
* далее лексический анализатор анализирует лексему, соответствующую данному слову, и выполняет действия, описанные для данной лексемы;
* формируется протокол работы.

Распознавание цепочек основывается на работе конечных автоматов. Работу конечного автомата можно проиллюстрировать с помощью графа переходов. Пример графа для цепочки “char” представлен на рисунке 3.5, где S0 – начальное, а S4 – конечное состояние автомата.

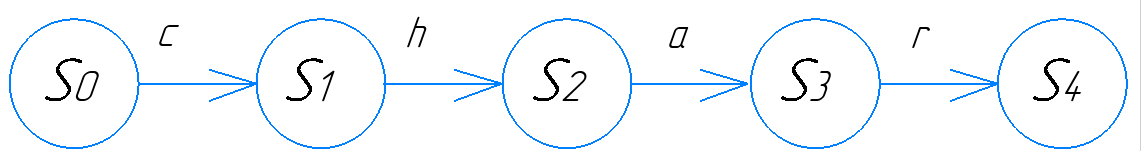


Рисунок 3.5 Пример графа переходов для цепочки char

На рисунке 3.6 представлен листинг графа переходов для цепочки char.

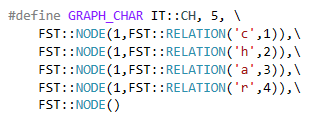


Рисунок 3.6 Граф переходов для цепочки char

**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.

* + - 1. **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 AEV-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 AEV-2020, представлен в таблице 4.1.

Таблица 4.1 – Перечень правил грамматики языка AEV-2020

|  |  |  |
| --- | --- | --- |
| Нетерминальный символ | Цепочки правил | Какие правила порождает |
| S-> | cfi(P){K}S  nfi(P){K}S  cfi(P){K}  nfi(P){K}  m{K} | Правила, описывающие общую структуру программы |
| P-> | si,P  si  ni,P  ni | Правила, описывающие параметры функции |
| K-> | tni;K  tci;K  tni=E;K  tci=E;K  i=E;K  oE;K  ri;  rl;  tni;  tci;  tni=E;  tci=E;  i=E;  oE;  u(iLi){E}K  u(iLi){E}  u(iLl){E}K  u(iLl){E} | Правила, описывающие основные синтаксические конструкции |
| E-> | i  iE  i;  i;E  i=iAE  i=iA  l  lE  l;  iA  lA  (W)  (W)A | Правила, описывающие выражения |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Нетерминальный символ | Цепочки правил | Какие правила порождает |
| E-> | (E)A  i()  i()A  p(W)  p(W)A  a(W)  a(W)A  o(i);E  o(i); | Правила, описывающие выражения |
| W-> | i  l  i,W  l,W | Правила, описывающие фактические параметры функции |
| A-> | +E  -E  \*E  /E | Правила для арифметических знаков |
| L-> | <  >  !  &  [  ] | Правила для логических знаков |
| C-> | i=E;C  oi;C  ol;C  i(W);C  i=E;  oi;  ol;  i(W); | Правила для тела цикла |

**4.3 Построение конечного магазинного автомата**

Формальное описание МП-автомата:



 - множество состояний;

 - алфавит входных символов;

 - специальный алфавит магазинных символов;

-функция переходов автомата , где  - множество подмножеств ;

 - начальное состояние автомата;

- начальное состояние магазина (маркер дна);

- множество конечных состояний.

Конфигурация (текущее состояние автомата) описывается тройкой , где - текущее состояние автомата, - остаток цепочки,  - цепочка-содержимое магазина.

Начальное состояние , - начальное состояние автомата,  - входная цепочка, - маркер дна магазина.

Цепочка  является допустимой (распознается) автоматом , если  и .

Работа автомата 

1. состояние автомата 
2. читает символ  находящийся под головкой (сдвигает ленту);
3. не читает ничего (читает , не сдвигает ленту);
4. из  определяет новое состояние , если  или .
5. читает верхний (в стеке) символ  и записывает цепочку  т.к. , при этом, если , то верхний символ магазина просто удаляется.

Работа автомата заканчивается 

Схема работы автомата с магазинной памятью представлена на рисунке 4.2.

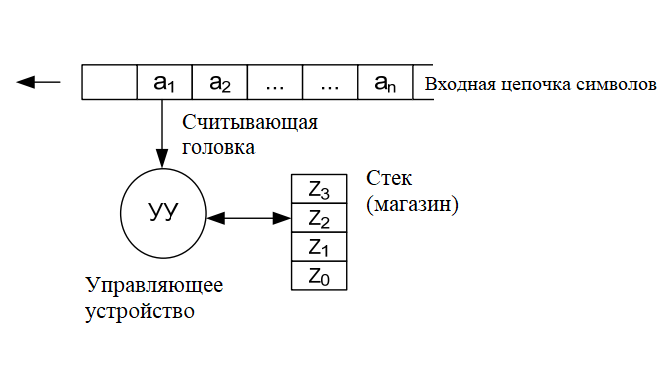
****

Рисунок 4.2 Схема автомата с магазинной памятью

**4.4 Основные структуры данных**

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

**4.5 Описание алгоритма синтаксического разбора**

Алгоритма работы синтаксического анализа можно описать следующим образом:

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

**4.6 Структура и перечень сообщений синтаксического анализатора**

Перечень сообщений синтаксического анализатора языка программирования AEV-2020 представлен в таблице 4.2.

Таблица 4.2 – Сообщения синтаксического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 100 | Неверная структура программы |
| 101 | Ошибка в параметрах функции |
| 102 | Неверная синтаксическая конструкция |
| 103 | Ошибка в выражении |
| 104 | Ошибка в параметрах вызываемой функции |
| 105 | Ошибка в построении арифметического выражения |
| 106 | Ошибка в построении логического выражения |
| 107 | Ошибка в теле цикла |

**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**

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, приведенные 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**

Результатом синтаксического разбора исходного кода на языке программирования AEV-2020 из контрольного примера, представленного в приложении А, является трассировка и дерево разбора, представленные в приложении Д.

* + - 1. **Разработка семантического анализатора**

**5.1 Структура семантического анализатора**

Семантический анализатор – часть транслятора, выполняющая cемантический анализ, то есть исходный код проверяется на наличие ошибок, которые невозможно отследить при помощи регулярной и контекстно-свободной грамматики. На вход принимаются результаты работ лексического и синтаксического анализаторов, то есть таблицы лексем, идентификаторов, и результат работы синтаксического анализатора, то есть дерево разбора. Некоторые проверки (такие как проверка на единственность точки входа, проверка на предварительное объявление переменной) осуществляются в процессе лексического анализа.

Семантический анализатор в языке программирования AEV-2020 не выделен в отдельную фазу.

Структура семантического анализатора представлена на рисунке 5.1.



Рисунок 5.1 Структура семантического анализатора

**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 AEV-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 |
| 300 | Превышено максимальное количество параметров при вызове функции |
| 301 | Попытка переопределения идентификатора |
| 302 | Отсутствует объявление идентификатора |
| 303 | Отсутствует точка входа main |
| 304 | Точка входа main должна быть единственной |
| 305 | Несоответствие типов |
| 306 | Запрещено присваивать значение функции |
| 307 | Запрещено делить на ноль |
| 308 | Использование стандартных функций без подключения стандартной библиотеки |
| 309 | Попытка подключить стандартную библиотеку более одного раза |
| 310 | Попытка проведения операций со строковым типом данных |
| 311 | Тип возвращаемого значения и тип функции не совпадают |
| 312 | Неправильное количество параметров при вызове функции |
| 313 | Неправильный тип параметров при вызове функции |
| 314 | Превышено допустимое количество параметров функции |
| 315 | Неправильный вызов функции |

**5.4 Принцип обработки ошибок**

При обнаружении хотя бы одной семантической ошибки транслятор прекратит свою работу.

**5.5 Контрольный пример**

Контрольный пример заключается в тестировании функций семантического анализатора при наличии соответствующих ошибок в исходном коде. Ошибки, диагностируемые семантическим анализатором:

1. Код программы:

main {

num a;

init num b;

a = 2;

b = 12;

out(a);

out(b);

}

Генерируемая ошибка представлена на рисунке 5.2



Рисунок 5.2 Сообщение об ошибке в коде под пунктом 1

1. Код программы:

num function f(num x, num y) {

init num a;

init num b;

a = 2;

b = 12;

out(a);

out(b);

}

Генерируемая ошибка представлена на рисунке 5.3



Рисунок 5.3 Сообщение об ошибки в коде под пунктом 2

1. Код программы:

main {

init num a;

a = 2;

out(a);

}

main() {

init num a=1;

init num b =5;

init num c=a+b;

out(c);

}

Генерируемая ошибка представлена на рисунке 5.3



Рисунок 5.4 Сообщение об ошибке в коде под пунктом 3

* + - 1. **Вычисление выражений**
  1. **Выражения, допускаемые языком**

В языке программирования AEV-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) |

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

Все выражения языка AEV-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.

## **Программная реализация обработки выражений**

Программная реализация преобразования выражений к польской записи представлена в приложении Е.

* 1. **Контрольный пример**

Результаты преобразования выражений исходного кода в польскую запись представлены в приложении Е. Пример преобразования выражения в контрольном примере в обратную польскую запись представлен в таблице 6.2.

Таблица 6.2 – Преобразование выражения в обратную польскую запись

|  |  |  |
| --- | --- | --- |
| Выражение | Стек | Результат |
| 5\*(b-a)+6/2 |  |  |
| \*(b-a)+6/2 |  | 5 |
| (b-a)+6/2 | \* | 5 |
| b-a)+6/2 | \*( | 5 |
| -a)+6/2 | \*( | 5b |
| a)+6/2 | \*(- | 5b |
| )+6/2 | \*(- | 5ba |
| +6/2 | \* | 5ba- |
| 6/2 | + | 5ba-\* |
| /2 | + | 5ba-\*6 |
| 2 | +/ | 5ba-\*6 |
|  | +/ | 5ba-\*62 |
|  |  | 5ba-\*62/+ |

* + - 1. **Генерация кода**
  1. **Структура генератора кода**

В языке программирования AEV-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 JavaScript, который будет являться результатом работы транслятора. В случае возникновения ошибок генерация кода не будет осуществляться. Структура генератора кода языка программирования AEV-2020 представлена на рисунке 7.1.

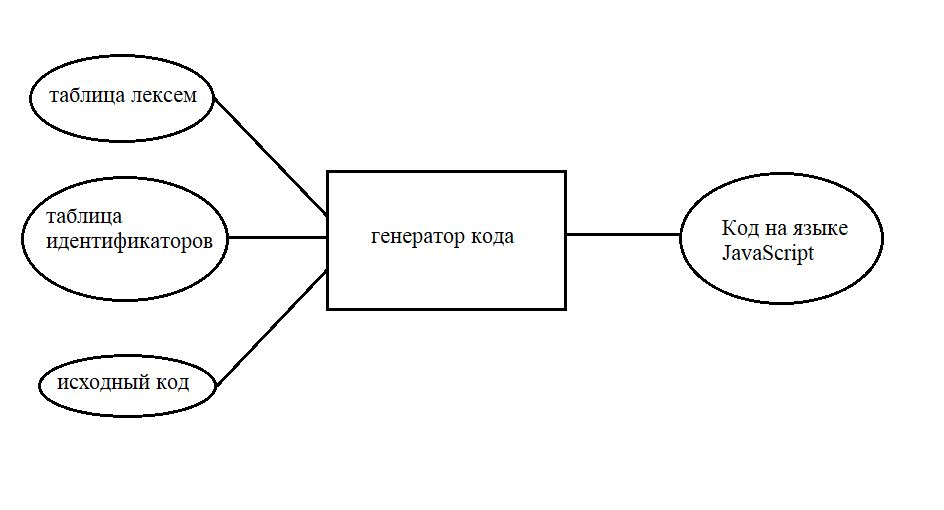


Рисунок 7.1 Структура генератора кода

* 1. **Представление типов данных в оперативной памяти**

Язык программирования AEV-2020 требует указывать тип данных при объявлении идентификатора с использованием ключевого слова init. В языке программирования JavaScript не обязательно указывать тип данных при объявлении. Объявление происходит с указанием ключевого слова var. Соответствия между типами данных идентификаторов языка программирования AEV-2020 и языка программирования JavaScript представлены в таблице 7.1.

Таблица 7.1 – Соответствие типов идентификаторов языка AEV-2020 и языка JavaScript

|  |  |
| --- | --- |
| Тип идентификаторов языка AEV-2020 | Тип данных языка C# |
| num (беззнаковый целый) | Number (числовой) |
| char (символьный) | String (строковый) |

* 1. **Статическая библиотека**

Для языка программирования AEV-2020 статическая библиотека как отдельный файл не реализована. Преобразование функций происходит на этапе генерации кода.

* 1. **Особенности алгоритма генерации кода**

Алгоритм генерации исходного кода на языке программирования AEV-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 трех лексем представлен на рисунке 7.2;
* когда каждая лексема из таблицы разобрана, код сгенерирован, файл закрывается;

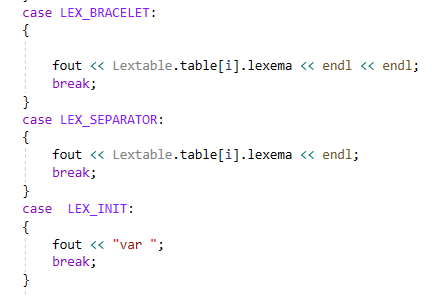


Рисунок 7.2 Часть листинга кода генерации в язык JavaScript

* 1. **Входные параметры генератора кода**

На этапе генерации кода транслятор языка программирования AEV-2020 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>.

* 1. **Контрольный пример**

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

1. **Тестирование транслятора**
   1. **Общие положения**

В результате обработки исходного кода программы, представленного в приложении А, транслятор языка программирования AEV-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.

* 1. **Результаты тестирования**

Результаты обработки транслятором исходного кода с допущенными ошибками приведены в таблице 8.1. В таблице приведены ошибок на разных этапах анализа: лексического, синтаксического, семантического. В квадратных скобках указано, на каком этапе анализа была сгенерирована ошибка.

Таблица 8.1 – Тестирование кода на языке программирования AEV-2020

|  |  |
| --- | --- |
| Код с ошибкой | Генерируемое сообщение |
| main  {  init char a='15';  out(8a);  } |  |
| num function Fun(num a,)  {  init char p;  p = 'y';  init num m=1;  return m;  }  main  {  init num a=15;  init num b = 3;  init num z;  z=a+b+;  } |  |

Продолжение таблицы 8.1

|  |  |
| --- | --- |
| Код с ошибкой | Генерируемое сообщение |
| main  {  init num a;  a = 2  out(a);  } |  |
| main  {  init num a;  a = 2/0;  out(a);  } |  |
| main  {  init num a;  a = 'l';  out(a);  } |  |

**Заключение**

В данном курсовом проекте был разработан транслятор и генератор кода для языка программирования AEV-2020 с минимальными требованиями. 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.

Окончательная версия языка программирования AEV-2020 включает:

* 2 типа данных;
* поддержка операции вывода;
* возможность вызова функций стандартной библиотеки;
* наличие четырех арифметических операторов для вычисления выражений;
* наличие шести логических операторов для использования их в условии цикла;
* оператор цикла;
* структурированная система для обработки ошибок пользователя;
* поддержка функций.

Основные характеристики транслятора PED-2018:

* Возможность обработки 3 входных параметров;
* Возможность обработки 37 ошибок;
* Реализация 16 конечных автоматов;
* Реализация 62 цепочки правил грамматики;
* Наличие порядка 2400 строк кода.

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

**Графический материал**

**Список использованных источников**

* + - 1. Краткая история кодировок от ASCII до UTF-8 [Электронный ресурс] – Режим доступа: http://popel-studio.com/blog/article/kratkaya-istoriya-kodirovok-ot-ascii-do-utf-8.html
      2. Ахо А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003. – 768с.
      3. Польская запись [Электронный ресурс] – Режим доступа: <https://studfiles.net/preview/2792990/>
      4. Прата, С. Язык программирования С++. Лекции и упражнения / С. Прата. – М., 2006 — 1104 c.

**Приложение А**

Контрольный пример

num function FunNum(num a, num b) {

init num m=1;

init num n;

n=4;

init num k=1;

until(n>m) {

k=k+a-b;

n=n-1;

}

return k;

}

char function FunCh() {

init char k='p';

return k;

}

main {

avr(2,3,4);

init num a=15;

init num b = 3;

init num z;

z=FunNum(a,b);

init char res;

res = FunCh();

init num result = 1;

result = pow(2,3);

init num p = 1;

p=(2\*(13+4)-(9+3)/2)\*3;

out(z);

out(result);

out(res);

out(p);

}

# **Приложение Б**

Конечные автоматы

#define N\_GRAPHS 16

#define GRAPH\_SEPARATORS 2,\

FST::NODE(18, FST::RELATION(';',1), FST::RELATION(',',1),\

FST::RELATION('}',1), FST::RELATION('{',1),\

FST::RELATION(')',1), FST::RELATION('(',1),\

FST::RELATION('+',1), FST::RELATION('\*',1),\

FST::RELATION('-',1), FST::RELATION('/',1),\

FST::RELATION('=',1), FST::RELATION('>',1),\

FST::RELATION('<',1), FST::RELATION(']',1),\

FST::RELATION('[',1), FST::RELATION('!',1),\

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

FST::NODE()

#define GRAPH\_UNTIL 6, \

FST::NODE(1,FST::RELATION('u',1)),\

FST::NODE(1,FST::RELATION('n',2)),\

FST::NODE(1,FST::RELATION('t',3)),\

FST::NODE(1,FST::RELATION('i',4)),\

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

FST::NODE()

#define GRAPH\_POW 4, \

FST::NODE(1,FST::RELATION('p',1)),\

FST::NODE(1,FST::RELATION('o',2)),\

FST::NODE(1,FST::RELATION('w',3)),\

FST::NODE()

#define GRAPH\_AVR 4,\

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

FST::NODE(1,FST::RELATION('v',2)),\

FST::NODE(1,FST::RELATION('r',3)),\

FST::NODE()

#define GRAPH\_FORWARD\_SLASH 2,\

FST::NODE(1,FST::RELATION('/',1)),\

FST::NODE()

#define GRAPH\_NUMLIT IT::NUM, 2, \

FST::NODE(10,FST::RELATION('0',1),FST::RELATION('1',1),\

FST::RELATION('3',1),FST::RELATION('2',1),\

FST::RELATION('5',1),FST::RELATION('4',1),\

FST::RELATION('7',1),FST::RELATION('6',1),\

FST::RELATION('8',1),FST::RELATION('9',1)),\

FST::NODE(10,FST::RELATION('0',1),FST::RELATION('1',1),\

FST::RELATION('3',1),FST::RELATION('2',1),\

FST::RELATION('5',1),FST::RELATION('4',1),\

FST::RELATION('7',1),FST::RELATION('6',1),\

FST::RELATION('8',1),FST::RELATION('9',1))

#define GRAPH\_CHARLIT IT::CH, 3,\

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

FST::NODE(142, \

FST::RELATION('A', 1), FST::RELATION('B', 1), \

FST::RELATION('C', 1), FST::RELATION('D', 1), \

FST::RELATION('E', 1), FST::RELATION('F', 1), \

FST::RELATION('G', 1), FST::RELATION('H', 1), \

FST::RELATION('I', 1), FST::RELATION('J', 1), \

FST::RELATION('K', 1), FST::RELATION('L', 1), \

FST::RELATION('M', 1), FST::RELATION('N', 1), \

FST::RELATION('O', 1), FST::RELATION('P', 1), \

FST::RELATION('Q', 1), FST::RELATION('R', 1), \

FST::RELATION('S', 1), FST::RELATION('T', 1), \

FST::RELATION('U', 1), FST::RELATION('V', 1), \

FST::RELATION('W', 1), FST::RELATION('X', 1), \

FST::RELATION('Y', 1), FST::RELATION('Z', 1), \

FST::RELATION('a', 1), FST::RELATION('b', 1), \

FST::RELATION('c', 1), FST::RELATION('d', 1), \

FST::RELATION('e', 1), FST::RELATION('f', 1), \

FST::RELATION('g', 1), FST::RELATION('h', 1), \

FST::RELATION('i', 1), FST::RELATION('j', 1), \

FST::RELATION('k', 1), FST::RELATION('l', 1), \

FST::RELATION('m', 1), FST::RELATION('n', 1), \

FST::RELATION('o', 1), FST::RELATION('p', 1), \

FST::RELATION('q', 1), FST::RELATION('r', 1), \

FST::RELATION('s', 1), FST::RELATION('t', 1), \

FST::RELATION('u', 1), FST::RELATION('v', 1), \

FST::RELATION('w', 1), FST::RELATION('x', 1), \

FST::RELATION('y', 1), FST::RELATION('z', 1), \

FST::RELATION('1', 1), FST::RELATION('2', 1), \

FST::RELATION('3', 1), FST::RELATION('4', 1), \

FST::RELATION('5', 1), FST::RELATION('6', 1), \

FST::RELATION('7', 1), FST::RELATION('8', 1), \

FST::RELATION('9', 1), FST::RELATION('0', 1), \

\

FST::RELATION('А', 1), FST::RELATION('Б', 1), \

FST::RELATION('В', 1), FST::RELATION('Г', 1), \

FST::RELATION('Д', 1), FST::RELATION('Е', 1), \

FST::RELATION('Ё', 1), FST::RELATION('Ж', 1), \

FST::RELATION('З', 1), FST::RELATION('И', 1), \

FST::RELATION('Й', 1), FST::RELATION('К', 1), \

FST::RELATION('Л', 1), FST::RELATION('М', 1), \

FST::RELATION('Н', 1), FST::RELATION('О', 1), \

FST::RELATION('П', 1), FST::RELATION('Р', 1), \

FST::RELATION('С', 1), FST::RELATION('Т', 1), \

FST::RELATION('У', 1), FST::RELATION('Ф', 1), \

FST::RELATION('Х', 1), FST::RELATION('Ц', 1), \

FST::RELATION('Ч', 1), FST::RELATION('Ш', 1), \

FST::RELATION('Щ', 1), FST::RELATION('Ъ', 1), \

FST::RELATION('Ы', 1), FST::RELATION('Ь', 1), \

FST::RELATION('Э', 1), FST::RELATION('Ю', 1), \

FST::RELATION('Я', 1), FST::RELATION('а', 1), \

FST::RELATION('б', 1), FST::RELATION('в', 1), \

FST::RELATION('г', 1), FST::RELATION('д', 1), \

FST::RELATION('е', 1), FST::RELATION('ё', 1), \

FST::RELATION('ж', 1), FST::RELATION('з', 1), \

FST::RELATION('и', 1), FST::RELATION('й', 1), \

FST::RELATION('к', 1), FST::RELATION('л', 1), \

FST::RELATION('м', 1), FST::RELATION('н', 1), \

FST::RELATION('о', 1), FST::RELATION('п', 1), \

FST::RELATION('р', 1), FST::RELATION('с', 1), \

FST::RELATION('т', 1), FST::RELATION('у', 1), \

FST::RELATION('ф', 1), FST::RELATION('х', 1), \

FST::RELATION('ц', 1), FST::RELATION('ч', 1), \

FST::RELATION('ш', 1), FST::RELATION('щ', 1), \

FST::RELATION('ъ', 1), FST::RELATION('ы', 1), \

FST::RELATION('ь', 1), FST::RELATION('э', 1), \

FST::RELATION('ю', 1), FST::RELATION('я', 1), \

FST::RELATION(' ', 1), FST::RELATION('.', 1), \

FST::RELATION(',', 1), FST::RELATION('?', 1), \

FST::RELATION('!', 1), FST::RELATION(';', 1), \

FST::RELATION(':', 1), FST::RELATION('-', 1), \

FST::RELATION(')', 1), FST::RELATION('(', 1), \

FST::RELATION('+', 1), FST::RELATION('\*', 1), \

FST::RELATION('/', 1), FST::RELATION('\'', 2)), \

FST::NODE()

#define GRAPH\_MAIN IT::NUM, 5, \

FST::NODE(1,FST::RELATION('m',1)),\

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

FST::NODE(1,FST::RELATION('i',3)),\

FST::NODE(1,FST::RELATION('n',4)),\

FST::NODE()

#define GRAPH\_INT IT::NUM, 4, \

FST::NODE(1,FST::RELATION('n',1)),\

FST::NODE(1,FST::RELATION('u',2)),\

FST::NODE(1,FST::RELATION('m',3)),\

FST::NODE()

#define GRAPH\_CHAR IT::CH, 5, \

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

FST::NODE(1,FST::RELATION('h',2)),\

FST::NODE(1,FST::RELATION('a',3)),\

FST::NODE(1,FST::RELATION('r',4)),\

FST::NODE()

#define GRAPH\_FUNCTION 9, \

FST::NODE(1,FST::RELATION('f',1)),\

FST::NODE(1,FST::RELATION('u',2)),\

FST::NODE(1,FST::RELATION('n',3)),\

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

FST::NODE(1,FST::RELATION('t',5)),\

FST::NODE(1,FST::RELATION('i',6)),\

FST::NODE(1,FST::RELATION('o',7)),\

FST::NODE(1,FST::RELATION('n',8)),\

FST::NODE()

#define GRAPH\_RETURN 7, \

FST::NODE(1,FST::RELATION('r',1)),\

FST::NODE(1,FST::RELATION('e',2)),\

FST::NODE(1,FST::RELATION('t',3)),\

FST::NODE(1,FST::RELATION('u',4)),\

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

FST::NODE(1,FST::RELATION('n',6)),\

FST::NODE()

#define GRAPH\_INIT 5, \

FST::NODE(1,FST::RELATION('i',1)),\

FST::NODE(1,FST::RELATION('n',2)),\

FST::NODE(1,FST::RELATION('i',3)),\

FST::NODE(1,FST::RELATION('t',4)),\

FST::NODE()

#define GRAPH\_OUT 4, \

FST::NODE(1, FST::RELATION('o', 1)),\

FST::NODE(1, FST::RELATION('u', 2)),\

FST::NODE(1, FST::RELATION('t', 3)),\

FST::NODE()

#define GRAPH\_ID 2, \

FST::NODE(51, \

FST::RELATION('A', 1), FST::RELATION('B', 1),\

FST::RELATION('C', 1), FST::RELATION('D', 1), \

FST::RELATION('E', 1), FST::RELATION('F', 1), \

FST::RELATION('G', 1), FST::RELATION('H', 1), \

FST::RELATION('I', 1), FST::RELATION('J', 1), \

FST::RELATION('K', 1), FST::RELATION('L', 1), \

FST::RELATION('M', 1), FST::RELATION('N', 1), \

FST::RELATION('O', 1), FST::RELATION('P', 1), \

FST::RELATION('Q', 1), FST::RELATION('R', 1), \

FST::RELATION('S', 1), FST::RELATION('T', 1), \

FST::RELATION('U', 1), FST::RELATION('V', 1), \

FST::RELATION('W', 1), FST::RELATION('X', 1), \

FST::RELATION('Y', 1), FST::RELATION('Z', 1), \

FST::RELATION('a', 1), FST::RELATION('b', 1), \

FST::RELATION('c', 1), FST::RELATION('d', 1), \

FST::RELATION('e', 1), FST::RELATION('f', 1), \

FST::RELATION('g', 1), FST::RELATION('h', 1),

FST::RELATION('j', 1), \

FST::RELATION('k', 1), FST::RELATION('l', 1), \

FST::RELATION('m', 1), FST::RELATION('n', 1), \

FST::RELATION('o', 1), FST::RELATION('p', 1), \

FST::RELATION('q', 1), FST::RELATION('r', 1), \

FST::RELATION('s', 1), FST::RELATION('t', 1), \

FST::RELATION('u', 1), FST::RELATION('v', 1), \

FST::RELATION('w', 1), FST::RELATION('x', 1), \

FST::RELATION('y', 1), FST::RELATION('z', 1)),\

FST::NODE(51, \

FST::RELATION('A', 1), FST::RELATION('B', 1), \

FST::RELATION('C', 1), FST::RELATION('D', 1), \

FST::RELATION('E', 1), FST::RELATION('F', 1), \

FST::RELATION('G', 1), FST::RELATION('H', 1), \

FST::RELATION('I', 1), FST::RELATION('J', 1), \

FST::RELATION('K', 1), FST::RELATION('L', 1), \

FST::RELATION('M', 1), FST::RELATION('N', 1), \

FST::RELATION('O', 1), FST::RELATION('P', 1), \

FST::RELATION('Q', 1), FST::RELATION('R', 1), \

FST::RELATION('S', 1), FST::RELATION('T', 1), \

FST::RELATION('U', 1), FST::RELATION('V', 1), \

FST::RELATION('W', 1), FST::RELATION('X', 1), \

FST::RELATION('Y', 1), FST::RELATION('Z', 1), \

FST::RELATION('a', 1), FST::RELATION('b', 1), \

FST::RELATION('c', 1), FST::RELATION('d', 1), \

FST::RELATION('e', 1), \

FST::RELATION('g', 1), FST::RELATION('h', 1), \

FST::RELATION('i', 1), FST::RELATION('j', 1), \

FST::RELATION('k', 1), FST::RELATION('l', 1), \

FST::RELATION('m', 1), FST::RELATION('n', 1), \

FST::RELATION('o', 1), FST::RELATION('p', 1), \

FST::RELATION('q', 1), FST::RELATION('r', 1), \

FST::RELATION('s', 1), FST::RELATION('t', 1), \

FST::RELATION('u', 1), FST::RELATION('v', 1), \

FST::RELATION('w', 1), FST::RELATION('x', 1), \

FST::RELATION('y', 1), FST::RELATION('z', 1))

#define GRAPH\_ERROR 2, \

FST::NODE(1, FST::RELATION('@', 1)),\

FST::NODE()

# **Приложение В**

Таблица лексем

00 nfi(ni,ni)

01 {

02 tni=l;

03 tni;

04 i=l;

05 tni=l;

06 u(i>i)

07 {

08 i=i+i-i;

09 i=i-l;

10 }

11 ri;

12 }

13 cfi()

14 {

15 tci=l;

16 ri;

17 }

19 m{

20 a(l,l,l);

21 tni=l;

22 tni=l;

23 tni;

24 i=i(i,i);

25 tci;

26 i=i();

27 tni=l;

28 i=p(l,l);

29 tni=l;

30 i=(l\*(l+l)-(l+l)/l)\*l;

31 o(i);

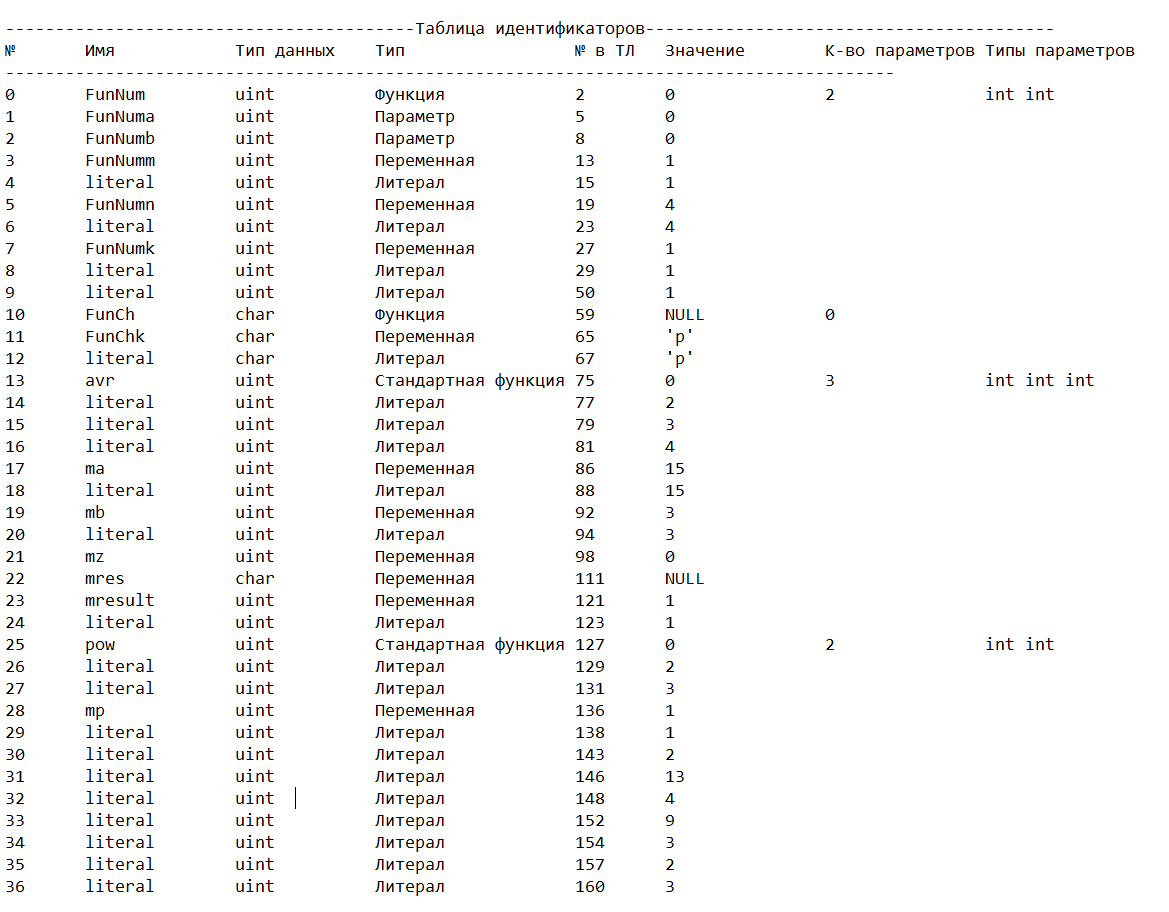
32 o(i);

33 o(i);

34 o(i);

35 }

Таблица идентификаторов



# **Приложение Г**

Структура данный грамматики Грейбах

Greibach greibach(NS('S'), TS('$'),

8,

Rule(NS('S'), GRB\_ERROR\_SERIES, 5, // структура программы

Rule::Chain(10,TS('c'),TS('f'),TS('i'),TS('('),NS('P'),TS(')'),TS('{'), NS('K'),TS('}'),NS('S')),

Rule::Chain(10, TS('n'),TS('f'),TS('i'),TS('('),NS('P'),TS(')'),TS('{'), NS('K'),TS('}'),NS('S')),

Rule::Chain(9,TS('c'),TS('f'),TS('i'),TS('('),TS(')'),TS('{'),NS('K'), TS('}'),NS('S')),

Rule::Chain(9,TS('n'),TS('f'),TS('i'),TS('('),TS(')'),TS('{'),NS('K'), TS('}'),NS('S')),

Rule::Chain(4,TS('m'),TS('{'),NS('K'),TS('}'))

),

Rule(NS('P'), GRB\_ERROR\_SERIES + 1, 4, //параметры ф-ии

Rule::Chain(4, TS('s'), TS('i'), TS(','), NS('P')),

Rule::Chain(2, TS('s'), TS('i')),

Rule::Chain(4, TS('n'), TS('i'), TS(','), NS('P')),

Rule::Chain(2, TS('n'), TS('i'))

),

Rule(NS('K'), GRB\_ERROR\_SERIES + 2, 19, // конструкции в функциях

Rule::Chain(5,TS('t'),TS('n'),TS('i'),TS(';'),NS('K')),

Rule::Chain(5,TS('t'),TS('c'),TS('i'),TS(';'),NS('K')),

Rule::Chain(7,TS('t'),TS('n'),TS('i'),TS('='),NS('E'),TS(';'),NS('K')),

Rule::Chain(7,TS('t'),TS('c'),TS('i'),TS('='),NS('E'),TS(';'),NS('K')),

Rule::Chain(5,TS('i'),TS('='),NS('E'),TS(';'),NS('K')),

Rule::Chain(4,TS('o'),NS('E'),TS(';'),NS('K')),

Rule::Chain(3,TS('r'),TS('i'),TS(';')),

Rule::Chain(3,TS('r'),TS('l'),TS(';')),

Rule::Chain(4,TS('t'),TS('n'),TS('i'),TS(';')),

Rule::Chain(4,TS('t'),TS('c'),TS('i'),TS(';')),

Rule::Chain(6,TS('t'),TS('n'),TS('i'),TS('='),NS('E'),TS(';')),

Rule::Chain(6,TS('t'),TS('c'),TS('i'),TS('='),NS('E'),TS(';')),

Rule::Chain(4,TS('i'),TS('='),NS('E'),TS(';')),

Rule::Chain(3,TS('o'),NS('E'),TS(';')),

Rule::Chain(6,TS('a'),TS('('),NS('W'),TS(')'),TS(';'),NS('K')),

Rule::Chain(10,TS('u'),TS('('),TS('i'),NS('L'),TS('i'),TS(')'),TS('{'), NS('E'), TS('}'), NS('K')),

Rule::Chain(9, TS('u'),TS('('),TS('i'),NS('L'), TS('i'), TS(')'), TS('{'), NS('E'), TS('}')),

Rule::Chain(10, TS('u'),TS('('),TS('i'),NS('L'), TS('l'), TS(')'), TS('{'), NS('E'), TS('}'), NS('K')),

Rule::Chain(9, TS('u'),TS('('),TS('i'),NS('L'), TS('l'), TS(')'), TS('{'), NS('E'), TS('}'))

),

Rule(NS('E'), GRB\_ERROR\_SERIES + 3, 22, //выражения

Rule::Chain(1, TS('i')),

Rule::Chain(2, TS('i'), NS('E')),

Rule::Chain(2, TS('i'), TS(';')),

Rule::Chain(3, TS('i'), TS(';'), NS('E')),

Rule::Chain(5, TS('i'), TS('='), TS('i'), NS('A'), NS('E')),

Rule::Chain(4, TS('i'), TS('='), TS('i'), NS('A')),

Rule::Chain(1, TS('l')),

Rule::Chain(2, TS('l'), NS('E')),

Rule::Chain(2, TS('l'), TS(';')),

Rule::Chain(2, TS('i'), NS('A')),

Rule::Chain(2, TS('l'), NS('A')),

Rule::Chain(3, TS('('), NS('W'), TS(')')),

Rule::Chain(4, TS('('), NS('W'), TS(')'), NS('A')),

Rule::Chain(4, TS('('), NS('E'), TS(')'), NS('A')),

Rule::Chain(3, TS('i'), TS('('), TS(')')),

Rule::Chain(4, TS('i'), TS('('), TS(')'), NS('A')),

Rule::Chain(4, TS('p'), TS('('), NS('W'), TS(')')),

Rule::Chain(5, TS('p'), TS('('), NS('W'), TS(')'), NS('A')),

Rule::Chain(4, TS('a'), TS('('), NS('W'), TS(')')),

Rule::Chain(5, TS('a'), TS('('), NS('W'), TS(')'), NS('A')),

Rule::Chain(6, TS('o'), TS('('), TS('i'), TS(')'), TS(';'), NS('E')),

Rule::Chain(5, TS('o'), TS('('), TS('i'), TS(')'), TS(';'))

),

Rule(NS('W'), GRB\_ERROR\_SERIES + 4, 4, //принимаемые параметры функции

Rule::Chain(1, TS('i')),

Rule::Chain(1, TS('l')),

Rule::Chain(3, TS('i'), TS(','), NS('W')),

Rule::Chain(3, TS('l'), TS(','), NS('W'))

),

Rule(NS('A'), GRB\_ERROR\_SERIES + 5, 4, // арифметические знаки

Rule::Chain(2, TS('+'), NS('E')),

Rule::Chain(2, TS('-'), NS('E')),

Rule::Chain(2, TS('\*'), NS('E')),

Rule::Chain(2, TS('/'), NS('E'))

),

Rule(NS('L'), GRB\_ERROR\_SERIES + 6, 6, // логические знаки

Rule::Chain(1, TS('<')),

Rule::Chain(1, TS('>')),

Rule::Chain(1, TS('!')),

Rule::Chain(1, TS('&')),

Rule::Chain(1, TS(']')),

Rule::Chain(1, TS('['))

),

Rule(NS('С'), GRB\_ERROR\_SERIES + 7, 8, // тело цикл

Rule::Chain(5, TS('i'), TS('='), NS('E'), TS(';'), NS('C')),

Rule::Chain(4, TS('o'), TS('i'), TS(';'), NS('C')),

Rule::Chain(4, TS('o'), TS('l'), TS(';'), NS('C')),

Rule::Chain(6, TS('i'), TS('('), NS('W'), TS(')'), TS(';'), NS('C')),

Rule::Chain(4, TS('i'), TS('='), NS('E'), TS(';')),

Rule::Chain(3, TS('o'), TS('i'), TS(';')),

Rule::Chain(3, TS('o'), TS('l'), TS(';')),

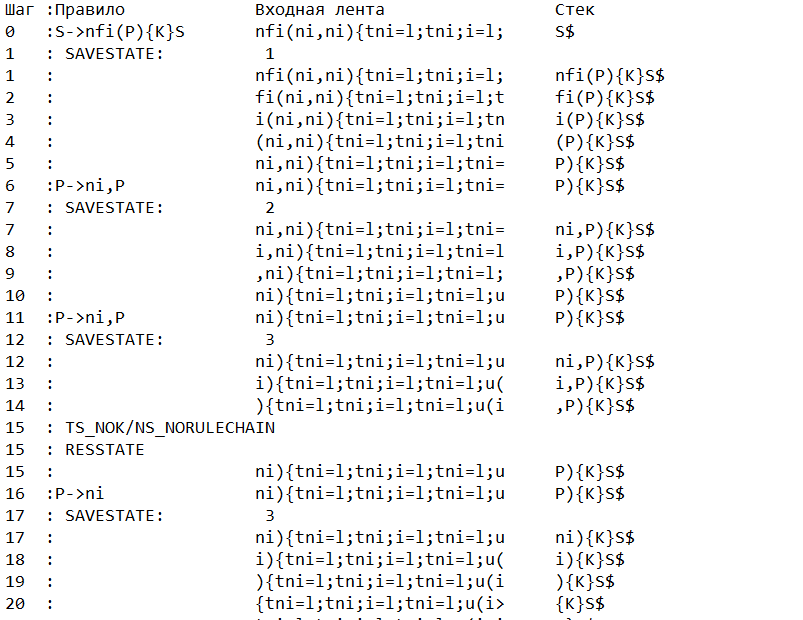
Rule::Chain(5, TS('i'), TS('('), NS('W'), TS(')'), TS(';'))

)

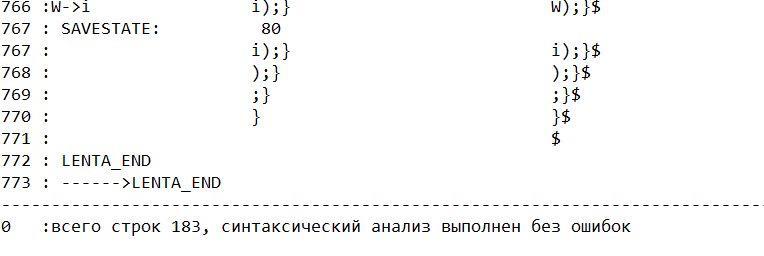
);

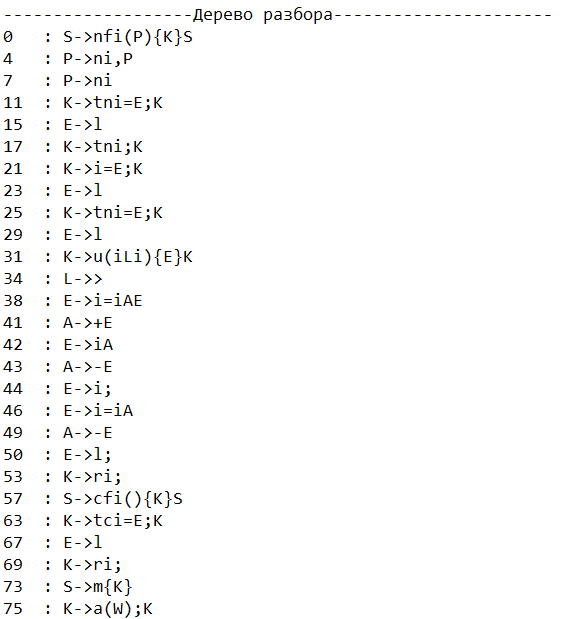
# **Приложение Д**

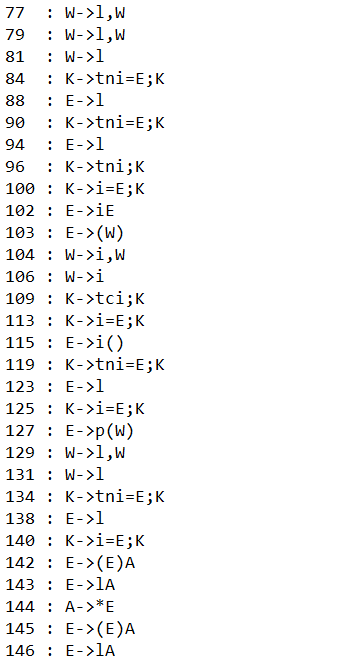
Начало трассировки

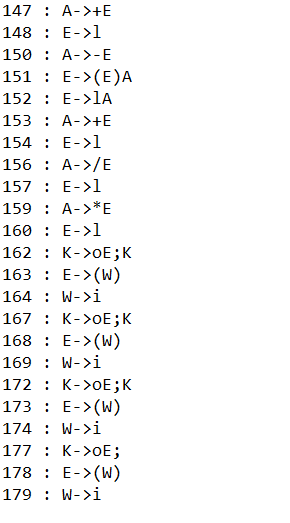


Конец трассировки









# **Приложение Е**

Преобразование исходного кода в польскую запись

00 nfi(ni,ni)

01 {

02 tni=l;

03 tni;

04 i=l;

05 tni=l;

06 u(i>i)

07 {

08 i=iii-+;

09 i=il-;

10 }

11 ri;

12 }

13 cfi()

14 {

15 tci=l;

16 ri;

17 }

19 m{

20 a(l,l,l);

21 tni=l;

22 tni=l;

23 tni;

24 i=i(i,i);

25 tci;

26 i=i();

27 tni=l;

28 i=p(l,l);

29 tni=l;

30 i=lll+\*ll+l/-l\*

31 o(i);

32 o(i);

33 o(i);

34 o(i);

35 }

# **Приложение Ж**

Код, сгенерированный в язык программирования JavaScript

function avr(a,b,c) {

var sum = 0;

var res = 1;

sum=a+b+c;

res=sum/3;

document.write(res); }

function FunNum(FunNuma, FunNumb) {

var FunNumm = 1;

var FunNumn;

FunNumn = 4;

var FunNumk = 1;

while(FunNumn > FunNumm) {

FunNumk = FunNumk + FunNuma - FunNumb;

FunNumn = FunNumn - 1; }

return FunNumk; }

function FunCh() {

var FunChk = 'p';

return FunChk; }

{ avr(2,3,4);

var ma = 15;

var mb = 3;

var mz;

mz = FunNum(ma, mb);

var mres;

mres = FunCh();

var mresult = 1;

mresult = Math.pow(2,3);

var mp = 1;

mp = (2 \* (13 + 4) - (9 + 3) / 2) \* 3;

document.write('<br>');

document.write(mz);

document.write('<br>');

document.write(mresult);

document.write('<br>');

document.write(mres);

document.write('<br>');

document.write(mp); }