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 1-40 01 01 «Программное обеспечение информационных технологий (программирование интернет-приложений)»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора HTA-2021»

Выполнил студент Хованский Тимофей Александрович

(Ф.И.О.)

Руководитель проекта асс. Мущук Артур Николаевич

(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.)

Консультант асс. Мущук Артур Николаевич

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

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

Нормоконтролер асс. Мущук Артур Николаевич

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

Курсовой проект защищен с оценкой

Минск 2021

**Содержание**

[Введение 5](#_Toc91057056)

[Глава 1. Спецификация языка программирования 6](#_Toc91057057)

[**1.1** **Характеристика языка программирования** 6](#_Toc91057058)

[**1.2** **Алфавит языка** 6](#_Toc91057059)

[**1.3** **Символы сепараторы** 6](#_Toc91057060)

[**1.4** **Применяемые кодировки** 7](#_Toc91057061)

[**1.5** **Типы данных** 7](#_Toc91057062)

[**1.6** **Преобразование типов данных** 8](#_Toc91057063)

[**1.7** **Идентификаторы** 8](#_Toc91057064)

[**1.8** **Литералы** 8](#_Toc91057065)

[**1.9** **Область видимости идентификаторов** 9](#_Toc91057066)

[**1.10** **Инициализация данных** 9](#_Toc91057067)

[**1.11** **Инструкции языка** 9](#_Toc91057068)

[**1.12** **Операции языка** 10](#_Toc91057069)

[**1.13** **Выражения и их вычисления** 10](#_Toc91057070)

[**1.14** **Программные конструкции языка** 11](#_Toc91057071)

[**1.15** **Область видимости** 11](#_Toc91057072)

[**1.16** **Семантические проверки** 11](#_Toc91057073)

[**1.17** **Распределение оперативной памяти на этапе выполнения** 12](#_Toc91057074)

[**1.18** **Стандартная библиотека и её состав** 12](#_Toc91057075)

[**1.19** **Ввод и вывод данных** 13](#_Toc91057076)

[**1.20** **Точка входа** 13](#_Toc91057077)

[**1.21** **Препроцессор** 13](#_Toc91057078)

[**1.22** **Соглашения о вызовах** 13](#_Toc91057079)

[**1.23** **Объектный код** 13](#_Toc91057080)

[**1.24** **Классификация сообщений транслятора** 13](#_Toc91057081)

[**1.25** **Контрольный пример** 14](#_Toc91057082)

[Глава 2. Структура транслятора 15](#_Toc91057083)

[**2.1 Компоненты транслятора, их назначение и принципы взаимодействия** 15](#_Toc91057084)

[**2.2 Перечень входных параметров** 16](#_Toc91057085)

[**2.3 Перечень протоколов, формируемых транслятором и их содержимое** 16](#_Toc91057086)

[Глава 3. Разработка лексического анализатора 17](#_Toc91057087)

[**3.1 Структура лексического анализатора** 17](#_Toc91057088)

[**3.2 Контроль входных символов** 17](#_Toc91057089)

[**3.3 Удаление избыточных символов** 18](#_Toc91057090)

[**3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов** 18](#_Toc91057091)

[**3.5 Основные структуры данных** 19](#_Toc91057092)

[**3.6 Принцип обработки ошибок** 20](#_Toc91057093)

[**3.7 Структура и перечень сообщений лексического анализатора** 20](#_Toc91057094)

[**3.8 Параметры лексического анализатора и режимы его работы** 21](#_Toc91057095)

[**3.9 Алгоритм лексического анализа** 21](#_Toc91057096)

[**3.10 Контрольный пример** 21](#_Toc91057097)

[Глава 4. Разработка синтаксического анализатора 22](#_Toc91057098)

[**4.1 Структура синтаксического анализатора** 22](#_Toc91057099)

[**4.2 Контекстно свободная грамматика, описывающая синтаксис языка** 22](#_Toc91057100)

[**4.3 Построение конечного магазинного автомата** 24](#_Toc91057101)

[**4.4 Основные структуры данных** 25](#_Toc91057102)

[**4.5 Описание алгоритма синтаксического разбора** 25](#_Toc91057103)

[**4.6 Структура и перечень сообщений синтаксического анализатора** 25](#_Toc91057104)

[**4.7 Параметры синтаксического анализатора и режимы его работы** 26](#_Toc91057105)

[**4.8 Принцип обработки ошибок** 26](#_Toc91057106)

[**4.9 Контрольный пример** 26](#_Toc91057107)

[Глава 5. Разработка семантического анализатора 27](#_Toc91057108)

[**5.1 Структура семантического анализатора** 27](#_Toc91057109)

[**5.2 Функции семантического анализатора** 27](#_Toc91057110)

[**5.3 Структура и перечень сообщений семантического анализатора** 27](#_Toc91057111)

[**5.4 Принцип обработки ошибок** 27](#_Toc91057112)

[**5.5 Контрольный пример** 28](#_Toc91057113)

[Глава 6. Преобразование выражений 29](#_Toc91057114)

[**6.1 Выражения, допускаемые языком** 29](#_Toc91057115)

[**6.2 Польская запись** 29](#_Toc91057116)

[**6.4 Контрольный пример** 30](#_Toc91057117)

[**7.2 Представление типов данных в оперативной памяти** 31](#_Toc91057118)

[**7.3 Статическая библиотека** 32](#_Toc91057119)

[**7.4** **Особенности алгоритма генерации кода** 32](#_Toc91057120)

[**7.5** **Входные параметры генератора кода** 33](#_Toc91057121)

[**Глава 8. Тестирование транслятора** 34](#_Toc91057122)

[**8.1 Тестирование фазы проверки на допустимость символов** 34](#_Toc91057123)

[**8.2 Тестирование лексического анализатора** 34](#_Toc91057124)

[**8.3 Тестирование синтаксического анализатора** 34](#_Toc91057125)

[**8.4 Тестирование семантического анализатора** 35](#_Toc91057126)

[Заключение 36](#_Toc91057127)

[Список использованной литературы 37](#_Toc91057128)

[Приложение А 38](#_Toc91057129)

[Приложение Б 39](#_Toc91057130)

[Приложение В 41](#_Toc91057131)

[Приложение Г 45](#_Toc91057132)

[Приложение Д 47](#_Toc91057133)

[Приложение Е 49](#_Toc91057134)

[Приложение Ж 51](#_Toc91057135)

[Приложение З 52](#_Toc91057136)

[52](#_Toc91057137)

# **Введение**

Задачей данного курсового проекта была поставлена разработка транслятора своего языка программирования HTA-2021. Он предназначен для выполнения простейших операций и арифметических действий над строками и числами.

Главной задачей транслятора заключается в том, чтобы сделать исходный код на данном языке программирования понятной компьютеру. Для решения этой задачи был выбран способ трансляции исходного кода моего языка программирования в исходный код на языке ассемблера. Язык ассемблера – это машинно-ориентированный язык, представляющий формат записи машинных команд, которые понятны для восприятия человеком.

Исходя из цели курсового проекта, были определены следующие задачи:

– разработка спецификации языка программирования;

– разработка структуры транслятора;

– разработка лексического анализатора;

– разработка синтаксического анализатора;

– разработка семантического анализатора;

– обработка выражений с помощью польской инверсии;

– генерация кода на язык ассемблера;

– тестирование транслятора;

Способы решения каждой задачи будут описаны в соответствующих главах курсового проекта.

В первой главе работы определена спецификация языка программирования.

Во второй главе представлена структура транслятора. В ней перечислены компоненты транслятора, их назначения и принципы взаимодействия.

В третьей главе описана разработка лексического анализатора, который создаёт таблицы лексем и идентификаторов.

В четвертой главе описана разработка синтаксического анализатора, который выполняет разбор исходного кода в соответствии с правилами языка программирования.

В пятой главе описан семантический анализатор, которые проверяет исходный код программы на наличие семантических ошибок.

В шестой главе описан способ преобразования выражений в польский формат.

В седьмой главе представлена генерация кода в язык ассемблера с помощью таблиц лексем и идентификаторов

В восьмой главе описывается тестирование транслятора

# **Глава 1. Спецификация языка программирования**

## **1.1 Характеристика языка программирования**

Язык HTA-2021 ­– это универсальный, строго типизированный, процедурный, не объектно-ориентированным, транслируемый.

## **1.2 Алфавит языка**

Алфавит языка программирования – набор символов, которые могут использоваться при написании исходного кода. Алфавит языка HTA-2021 основан на кодировке Windows-1251.Исходный код HTA-2021 может содержать символы латинского алфавита, цифры десятичной системы счисления от 0 до 9, русские символы разрешены только в строковых литералах, знаки препинания, знаки арифметических и логических операций.

## **1.3 Символы сепараторы**

Сепараторы служат для разделения операций языка. Сепараторы, применяемые в языке программирования HNA-2021, приведены в таблице 1.1.

Таблица 1.1 – Сепараторы

|  |  |
| --- | --- |
| Сепараторы | Назначение |
| ‘пробел’, ‘табуляция’,  ‘переход на новую строку’ | Разделяют входные лексемы |
| +, -, \*, /, % | Арифметические операторы. Используются в арифметических операциях |
| = | Оператор присваивания. Используется для присваивания значения переменной |
| <, >, ~, !, | Условные операторы. Используются для сравнения переменных и литералов |
| ( ) | Блок параметров функции, так же указывает приоритет в арифметических операциях |
| , | Разделяет параметры функции |
| { } | Ограничивают программные блоки инструкций |
| ; | Признак конца инструкции языка |
| “” | Ограничивают строковый литерала |
| ‘’ | Ограничивают символьный литерала |

## **1.4 Применяемые кодировки**

Для написания исходного кода на языке программирования HTA-2021 используется кодировка Windows-1251– набор символов и кодировка, являющаяся стандартной 8-битной кодировкой для русских версий Microsoft Windows – представленная на рисунке 1.1.

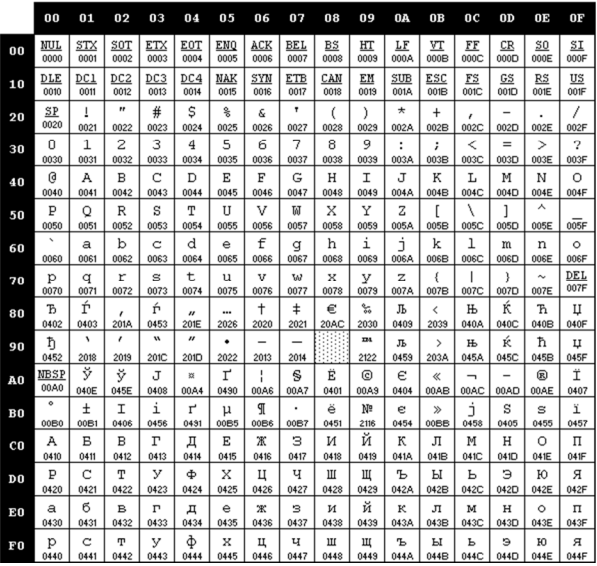


Рисунок 1.1 – Кодировка Windows-1251

## **1.5 Типы данных**

В языке HTA-2021 есть 3 типа данных: целочисленный, строковый и символьный. Описание типов данных, предусмотренных в данным языке представлено в таблице 1.2.

Таблица 1.2 – Типы данных

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| int | Фундаментальный тип данных, используемый для объявления целочисленных данных(4 байт). Инициализация по умолчанию: 0 |

Продолжение таблицы 1.2

|  |  |
| --- | --- |
| Charline | Фундаментальный тип данных. Предусмотрен для объявления строк. (1 символ – 1 байт). К строкам, переменным и литералам операции не применяются. Автоматическая инициализация строкой нулевой длины. Максимальное количество символов в строке – 255. |
| char | Фундаментальный тип данных, используемый для объявления символа. Символ в памяти занимает 1 байт. |

## **1.6 Преобразование типов данных**

В языке программирования HTA-2021 преобразование типов данных не поддерживается.

## **1.7 Идентификаторы**

Идентификаторы могут выступать в качестве имен функций, имен параметров, имен переменных. Зарезервированные идентификаторы не предусмотрены. Идентификаторы, объявленные внутри функционального блока, получают префикс, который отображается в таблице идентификаторов. В имени идентификатора допускаются символы латинского алфавита нижнего регистра и цифр. Идентификатор не может совпадать с ключевыми словами языка программирования. Максимальная длина идентификатора равна 20. При превышении длины идентификатора она будет урезаться**.**

## **1.8 Литералы**

Таблица 1.3 – Описание литералов

|  |  |
| --- | --- |
| Тип литерала | Описание |
| Литералы целого типа | Целочисленные литералы, только rvalue. Могут быть представлены в десятичной, двоичной и восьмеричной системах счисления. Для представления в системе необходимо сделать префикс двоичной(B) ((P||N)B),восьмеричной (O) ((P||N)O).Для обозначения знака перед числом ставится префик P для положительных и N для отрицательных чисел. |
| Строковые литералы | Символы, заключённые в “” (двойные кавычки), по умолчанию инициализируются пустой строкой. Литералы могут быть только rvalue. |

Продолжение таблицы 1.3

|  |  |
| --- | --- |
| Символьный литерал | Символ ,заключённый в ‘’. Литералы могут быть только rvalue. |

## **1.9 Область видимости идентификаторов**

Область видимости «сверху вниз» (по принципу С++). В языке HTA-2021 требуется обязательное объявление переменной перед её инициализацией и последующим использованием. Все переменные должны находиться внутри программного блока. Имеется возможность объявления одинаковых переменных в разных блоках, т. к. переменные, объявленные в одной функции, недоступны в другой. Каждая переменная получает постфикс – название функции, в которой она объявлена.

## **1.10 Инициализация данных**

Таблица 1.4 – Способы инициализации переменных

|  |  |
| --- | --- |
| Вид инициализации | Примечание |
| define <тип данных> <идентификатор>; | Автоматическая инициализация: переменные типа num инициализируются нулём, переменные типа string – пустой строкой. |
| define <тип данных> <идентификатор> = <выражение>; | Инициализация переменной с присваиванием значения. |

## **1.11 Инструкции языка**

Все возможные инструкции языка программирования HTA-2021 представлены в общем виде в таблице 1.5.

Таблица 1.5 – Инструкции языка программирования HTA-2021

|  |  |
| --- | --- |
| Инструкция | Запись на языке SA-2020 |
| Объявление переменной | define <тип данных> <идентификатор>;  define <тип данных> <идентификатор> = <выражение>; |
| Присваивание | <идентификатор> = <выражение >; |
| Объявление функции | <тип данных> Minor <идентификатор> (<тип данных> <идентификатор>, …) […] |
| Точка входа | MAJOR  {  …  } |

Продолжение таблицы 1.5

|  |  |
| --- | --- |
| Возврат значения из подпрограммы | give <идентификатор> | <литерал>; |
| Вывод данных | show (<идентификатор> | <литерал>);  showbr (<идентификатор> | <литерал>); |
| Условный оператор | if (<имя переменной, литерал><условный оператор><имя переменной, литерал>)  {  …  }  nif  {  …  };  Блок nif не обязателен. |
| Оператор цикла | Continue (<имя переменной, литерал><условный оператор><имя переменной, литерал>)  {  …  } |

## **1.12 Операции языка**

Язык программирования HTA-2021 может выполнять арифметические операции, представленные в таблице 1.6.

Таблица 1.6 – Операции и их приоритеты

|  |  |
| --- | --- |
| Операция | Приоритет операции |
| (  )  , | 1 |
| +  - | 2 |
| \*  / (деление с остатком)  % (деление по модулю) | 3 |

Максимальным значением приоритетности является “3”, минимальным “1” соответственно.

## **1.13 Выражения и их вычисления**

Выражение языка программирования HTA-2021 – это совокупность переменных, литералов, вызовов функций, знаков операций, скобок, которая может быть вычислена в соответствии с синтаксисом языка. В выражении должны участвовать операторы и операнды одного типа, а также функции, возвращающие значения того же типа. Круглые скобки в выражении используются для изменения приоритетов операций. Не допускается запись двух подряд арифметических операций. Также круглые скобки могут использоваться для передачи параметров функций. Фигурные скобки содержат блоки кода функций и циклов.

## **1.14 Программные конструкции языка**

Ключевые программные конструкции языка программирования HTA-2021 представлены чуть ниже в таблице 1.7.

Таблица 1.7 – Программные конструкции языка HTA-2021

|  |  |
| --- | --- |
| Конструкция | Описание |
| Главная функция (точка входа в приложение) | MAJOR  {  …  }; |
| Функция | <тип> Minor <идентификатор>([<тип> <идентификатор>]>][ <тип данных> <идентификатор>,]\*)  {  …  give <выражение>;  }; |
| Процедуры | process <идентификатор>(<тип><индефикатор>  {  …  }; |

Объявление функции или процедуры допустимо только перед точкой входа в программу, так как иначе функции не будут входить в область видимости программы.

## **1.15 Область видимости**

В языке HTA-2021 переменные обязаны обязаны быть объявленными внутри какой-либо функции. Глобальных переменных нет. Объявление пользовательских областей видимости не предусмотрено.

## **1.16 Семантические проверки**

Таблица с перечнем семантических проверок, предусмотренных языком HTA-2021, приведена в таблице 1.8.

Таблица 1.8 – Семантические проверки

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Соответствие типов в выражениях |
| 2 | Идентификатор должен быть объявлен до его использования |
| 3 | Операнды в арифметическом выражении не могут быть разных типов |
| 4 | Каждый идентификатор может быть объявлен только один раз |
| 5 | Проверка на превышение максимального размера строкового и целочисленного литералов |
| 6 | Соответствие типа возвращаемого значения с типом функции |

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

Все переменные размещаются в стеке. Таблицы лексем и идентификаторов размещены в структуры с выделенной под них оперативной памятью, которая очищается по окончанию работы транслятора.

## **1.18 Стандартная библиотека и её состав**

В языке HTA-2021 присутствует стандартная библиотека, которая подключается автоматически при трансляции исходного кода в язык ассемблера. Содержимое библиотеки и описание функций представлено в таблице 1.8.

Таблица 1.8 Стандартная библиотека языка HTA-2021

|  |  |
| --- | --- |
| Функция | Описание |
| int findSymb(string str,char ch) | Целочисленная функция.Возращает номер первого вхождения символа ch в строку str ( -1 в случае невхождения). |
| Int destiny(int a,int b) | Целочисленная функция. Генерерирует случайное число в диапазоне [a;b] |

Стандартная библиотека написана на языке С++, подключается к транслированному коду на этапе генерации кода. Вызовы стандартных функций доступны там же, где и вызов пользовательских функций. Также в стандартной библиотеке реализованы функции для манипулирования выводом, недоступные конечному пользователю. Для вывода предусмотрен оператор **show и showbr**. Эти функции представлены в таблице 1.9.

Таблица 1.9 Дополнительные функции стандартной библиотеки

|  |  |
| --- | --- |
| Функция на языке С++ | Описание |
| void OutputInt(int value) | Функция для вывода в стандартный поток значения целочисленного идентификатора/литерала. |

Продолжение таблицы 1.9

|  |  |
| --- | --- |
| void OutputStr(char\* line) | Функция для вывода в стандартный поток значения строкового идентификатора/литерала. |
| void ShowChar(char ch) | Функция для вывода в стандартный поток значения символьного идентификатора/литерала |

## **1.19 Ввод и вывод данных**

Ввод данных не поддерживается языком программирования HTA-2021.

showbr || show <идентификатор или литерал>; – вывод в стандартный поток вывода.

## **1.20 Точка входа**

Точкой входа в языке программирования HTA-2021 является функция MAJOR.

## **1.21 Препроцессор**

Препроцессор в языке программирования HTA-2021 не предусмотрен.

## **1.22 Соглашения о вызовах**

В языке вызов функций происходит по соглашению 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**

Язык транслируется в язык ассемблера.

## **1.24 Классификация сообщений транслятора**

В случае возникновения ошибки в исходном коде программы на языке программирования HTA-2021 и выявлении её транслятором в файл протокола выводится сообщение. Классификация обрабатываемых ошибок приведена в таблице 1.9.

Таблица 1.10 – Классификация сообщений транслятора

|  |  |
| --- | --- |
| Интервал | Описание ошибок |
| 0-109 | Системные ошибки |

Продолжение таблицы 1.10

|  |  |
| --- | --- |
| 200-299 | Ошибки лексического анализа |
| 600-699 | Ошибки синтаксического анализа |
| 300-399 | Ошибки семантического анализа |

## **1.25 Контрольный пример**

Контрольные примеры представлены в приложении А.

# **Глава 2. Структура транслятора**

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

Входными данными для транслятора является исходный код, написанный на языке программирования HTA-2021. Выходными данными является объектный код, а также протоколы работы транслятора, описанные в пункте 2.3. Компонентами транслятора являются лексический, синтаксический и семантический анализаторы, а также генератор кода на языке ассемблера. Принцип взаимодействия представлен на рисунке 2.1.



Рисунок 2.1 – Структура транслятора

Лексический анализ – первая фаза трансляции. Назначением лексического анализатора является нахождение ошибок лексики языка и формирование таблицы лексем и таблицы идентификаторов. Подробнее описан в 3 главе.

Синтаксический анализ – это основная часть транслятора, c для распознавания синтаксических конструкций. Входным параметром для синтаксического анализа является таблица лексем. Синтаксический анализатор распознаёт синтаксические конструкции, выявляет синтаксические ошибки при их наличии и формирует дерево разбора. При синтаксически правильном построении программы осуществляется переход к следующему этапу трансляции, иначе, работа транслятора останавливается. Подробнее рассмотрен в главе 4.

Семантический анализ в свою очередь является проверкой исходной программы на семантическую согласованность с определением языка, т.е. проверяет правильность текста исходной программы с точки зрения семантики. Подробное описание представлено в 5 главе.

Генератор кода – этап транслятора, выполняющий генерацию ассемблерного кода на основе полученных данных на предыдущих этапах трансляции.

Генератор кода принимает на вход таблицы идентификаторов и лексем и транслирует код на языке HTA-2021, прошедший все предыдущие этапы, в код на языке Ассемблера. Более полно описан в главе 7.

## **2.2 Перечень входных параметров**

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

Таблица 2.1 – Входные параметры транслятора языка HTA-2021

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<имя\_файла> | Входной файл с расширением .txt, в котором содержится исходный код на HTA-2021 | Не предусмотрено |
| -log:<имя\_файла> | Файл для записи результата работы транслятора | <имя\_файла>.log |
| -out:<имя\_файла> | Файл для записи результата работы транслятора | <имя\_файла>.out.asm |

Обязательным является только параметр -in, остальные, при отсутсвии соответствующего ключа, будут формироваться из имени файла с исходным кодом.

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

Таблица с перечнем протоколов, формируемых транслятором языка HTA-2021 и их назначением представлена в таблице 2.2

Таблица 2.2 – Протоколы, формируемые транслятором языка HTA-2021

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |
| Файл журнала, заданный параметром "-log:" | Файл с протоколом работы транслятора языка программирования HTA-2021. Содержит таблицу лексем и таблицу идентификаторов, протокол работы синтаксического анализатора и дерево разбора, полученные на этапе лексического и синтаксического анализа, а также результат работы алгоритма преобразования выражений к польской записи. |
| Выходной файл, c расширением ".asm" | Результат работы программы – файл, содержащий исходный код на языке ассемблера. |

При отсутствии соответствующих ключей файлы протоколов создаются автоматически.

# **Глава 3. Разработка лексического анализатора**

## **3.1 Структура лексического анализатора**

Лексический анализатор – часть транслятора, выполняющая лексический анализ. Лексический анализатор принимает обработанный и разбитый на отдельные компоненты исходный код на языке HTA-2021. На выходе формируется таблица лексем и таблица идентификаторов. Примеры лексических единиц: идентификаторы, числа, символы операций, служебные слова и т.д. 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. Структура лексического анализатора представлена на рисунке 3.1.



Рисунок 3.1 Структура лексического анализатора HTA-2021

## **3.2 Контроль входных символов**

Таблица для контроля входных символов представлена на рисунке 3.2.

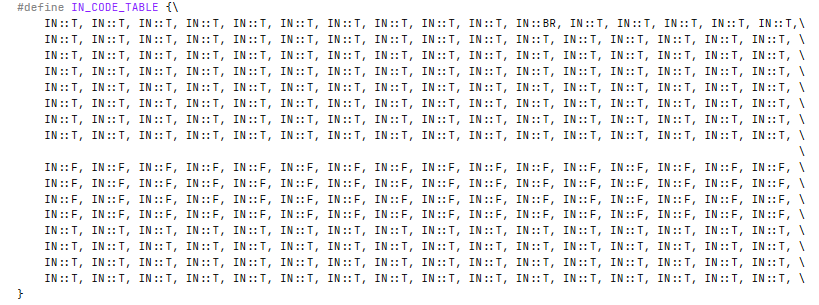


Рисунок 3.2. Таблица контроля входных символов

Таблица допустимости сформирована на основе кодировки Windows-1251. Она предназначена для проверки входных символов на допустимость. Символы представлены в шестнадцатеричной системе счисления. Каждому символу в данной таблице соответствует некое числовое значение: «T»,«F»,«I»,«BR».

«T» - разрешенные алфавитом символы (также разрешенными являются все символы кроме «F» и «I»);

«F» - запрещенные алфавитом символы;

«I» - символы, которые игнорируются;

«BR» - символ перевода каретки на новую строку;

## **3.3 Удаление избыточных символов**

Избыточными символами являются символы табуляции, пробелы и переходы на новую строку. Избыточные символы удаляются на этапе разбиения исходного кода на лексемы.

Описание алгоритма удаления избыточных символов:

1. Посимвольно считываем файл с исходным кодом программы.
2. Встреча пробела или знака табуляции является своего рода встречей символа-сепаратора.
3. В отличие от других символов-сепараторов не записываем в очередь лексем эти символы, т.е. игнорируем.

## **3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов**

Лексемы – это символы, соответствующие ключевым словам, символам операций и сепараторам, необходимые для упрощения дальнейшей обработки исходного кода программы. Каждому ключевому слову также соответствует автомат, по которому происходит разбор выражения. Автомат является детерминированным, то есть имеет конечное состояние. Проверка происходит следующим образом: на каждый автомат в массиве подаётся ключевое слово и с помощью графа переходов происходит разбор. Если разбор выполнен, то происходит заполнение таблицы лексем и при необходимости таблицы идентификаторов.

Таблица 3.1 Соответствие цепочек с лексемами

|  |  |  |
| --- | --- | --- |
| Тип цепочки | Цепочка | Лексема |
| Ключевые слова | define | d |
| int, char, charline | t |
| MAJOR | m |
| Minor | f |
| void | p |
| give | g |
| show | o |
| showbr | b |
| continue | u |
| if | w |
| nif | ! |
| Данные | Лексема | l |
| Идентификатор | i |
| Сепараторы | ; | ; |
| , | , |
| { | { |
| } | } |
| ( | ( |
| ) | ) |
| Операции | Арифметические (+,-,/,\*,%) и логические (>, <, ~, !) | v |
| Присваивания (=) | = |

Пример реализации таблицы лексем представлен в приложении Б.

Также в приложении В находятся конечные автоматы ключевых слов.

## **3.5 Основные структуры данных**

Основная структура данных, используемая для хранения таблицы идентификаторов представлена на рисунке 3.1.

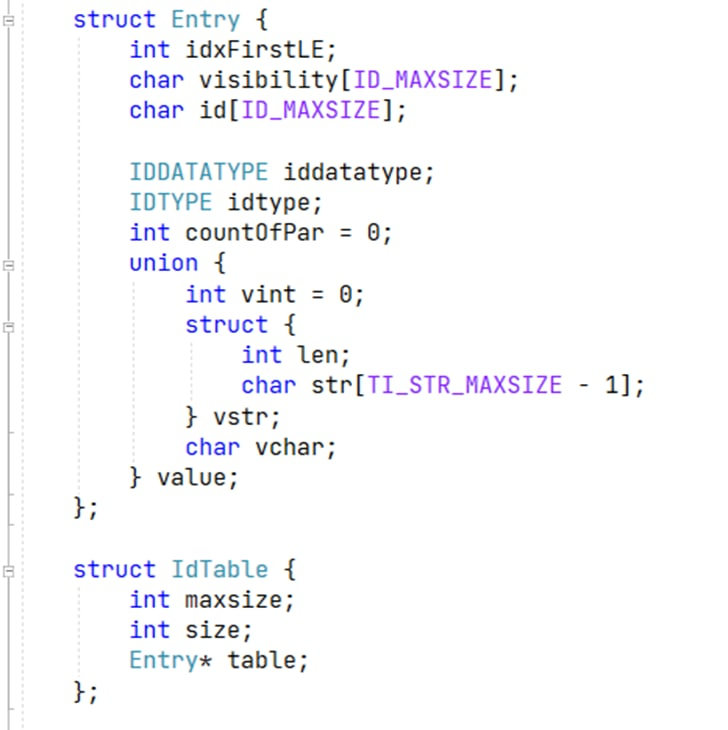


Рисунок 3.1. — Структура таблицы идентификаторов

Основная структура данных, используемая для хранения таблицы лексем представлена на рисунке 3.2.

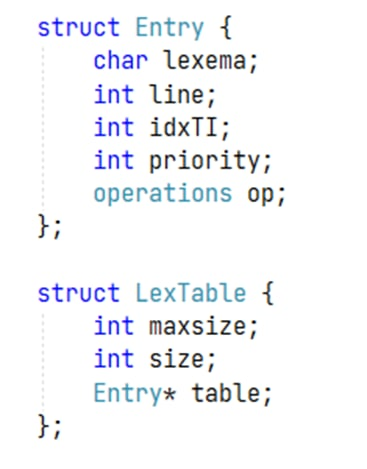


Рисунок 3.2. — Структура таблицы лексем

## **3.6 Принцип обработки ошибок**

При нахождении ошибок, транслятор пытается продолжить свою работу до завершения текущего этапа, чтобы вывести набор ошибок. При нахождении критической ошибки работа транслятора прекращается. Подсчет количества ошибок не ведется.

## **3.7 Структура и перечень сообщений лексического анализатора**

Перечень сообщений лексического анализатора представлен на рисунке 3.4.

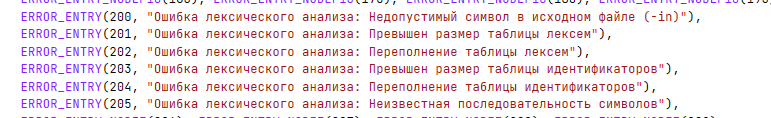


Рисунок 3.4 – Перечень ошибок лексического анализатора

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

Входным параметром функции лексического анализатора является исходный текст программы, написанный на языке HTA-2021, а также файл протокола.

## **3.9 Алгоритм лексического анализа**

Лексический анализ выполняется программой (входящей в состав транслятора), называемой лексическим анализатором. Цель лексического анализа — выделение и классификация лексем в тексте исходной программы. Лексический анализатор распознаёт и разбирает цепочки исходного текста программы. Этот разбор основывается на работе конечных автоматов, которую можно представить в виде графов.

Пример. Конечный автомат для распознавания ключевого слова main приведен на рисунке 3.5 S0 – начальное состояние, S4 – конечное состояние автомата.

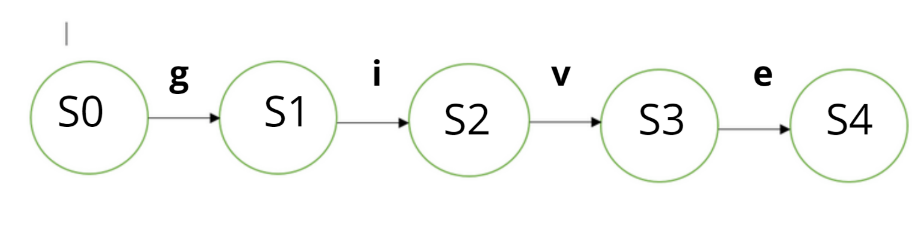


Рисунок 3.3 — Граф переходов для цепочки “give”

## **3.10 Контрольный пример**

Результат работы лексического анализатора – вывод в протокол таблицы лексем и идентификаторов – представлен в приложении Б.

# **Глава 4. Разработка синтаксического анализатора**

## **4.1 Структура синтаксического анализатора**

Синтаксический анализ – это фаза трансляции, выполняемая после лексического анализа и предназначенная для распознавания синтаксических конструкций. Входом для синтаксического анализа является таблица лексем и таблица идентификаторов, полученные после фазы лексического анализа. Выходом – дерево разбора.

Структура синтаксического анализатора представлена на рисунке 4.1.



Рисунок 4.1 – Структура синтаксического анализатора

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

В синтаксическом анализаторе транслятора языка HTA-2021 используется контекстно-свободная грамматика , где

T – множество терминальных символов (было описано в разделе 1.2 данной пояснительной записки),

N – множество нетерминальных символов (первый столбец таблицы 4.1),

P – множество правил языка (второй столбец таблицы 4.1),

S – начальный символ грамматики, являющийся нетерминалом.

Эта грамматика имеет нормальную форму Грейбах, т.к. она не леворекурсивная (не содержит леворекурсивных правил) и правила  имеют вид:

1. , где ; (или , или )
2. , где — начальный символ, при этом если такое правило существует, то нетерминал  не встречается в правой части правил.

Правила языка HTA-2021 представлены в приложении Г.

TS – терминальные символы, которыми являются сепараторы, знаки арифметических операций и некоторые строчные буквы.

NS – нетерминальные символы, представленные несколькими заглавными буквами латинского алфавита.

Таблица 4.1 – Перечень правил, составляющих грамматику языка и описание нетерминальных символов HTA-2021

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| S | tfiFBS  piFUS  m{N} | Стартовые правила, описывающее общую структуру программы. |
| F | (P)  () | Проверка наличия параметров функции |
| P | ti  ti,P | Проверка на правильность параметров функции при её объявлении |
| B | {Nri;}  {Nrl;}  {gi;}  {gl;} | Проверка наличия тела функции |
| E | i  l  (E)  iK  iM  lM  (E)M  iKM | Проверка на правильность арифметического выражения |
| W | i  l  i,W  l,W | Проверка на правильность параметров вызываемой функции |
| M | vE  vEM | Проверка на правильность арифметических действий |
| X | dti;X  dti=E;X  i=E;X  oi;X  ol;X  bi;X  bl;X  iK;X  dti;  dti=E;  i=E;  oi;X | Проверка на правильность конструкции в теле цикла или условного выражения |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
|  | ol;X  bi;X  bl;X  iK; |  |
| R | i  ivi  ivl  lvi | Проверка на правильность в условном выражении |
| K | (W)  () | Проверка на правильность вызова функции |

## **4.3 Построение конечного магазинного автомата**

Конечный автомат с магазинной памятью представляет собой семерку, описание которой представлено в таблице 4.2. Структура данного автомата показана в приложении Д.

Таблица 4.2 – Описание компонентов магазинного автомата

|  |  |  |
| --- | --- | --- |
| Компонента | Определение | Описание |
|  | Множество состояний автомата | Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата |
|  | Алфавит входных символов | Алфавит является множеством терминальных и нетерминальных символов, описание которых содержится в разделе 1.2 и в таблице 4.1. |
|  | Алфавит специальных магазинных символов | Алфавит магазинных символов содержит стартовый символ и маркер дна стека |
|  | Функция переходов автомата | Функция представляет из себя множество правил грамматики, описанных в таблице 4.1. |
|  | Начальное состояние автомата | Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики (нетерминальный символ А) |
|  | Начальное состояние магазина автомата | Символ маркера дна стека ($) |

Продолжение таблицы 4.2

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

## **4.4 Основные структуры данных**

Основные структуры данных и правила перехода, используемые на фазе синтаксического анализа, представлены в приложении Г.

## **4.5 Описание алгоритма синтаксического разбора**

Принцип работы автомата:

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

## **4.6 Структура и перечень сообщений синтаксического анализатора**

Перечень сообщений синтаксического анализатора представлен в листинге рисунке  4.1.

|  |
| --- |
| ERROR\_ENTRY(600, "Ошибка синтаксического анализа: Неправильная структура программы"),  ERROR\_ENTRY(601, "Ошибка синтаксического анализа: Отсутствует список параметров функции"),  ERROR\_ENTRY(602, "Ошибка синтаксического анализа: Ошибка в параметрах функции"),  ERROR\_ENTRY(603, "Ошибка синтаксического анализа: Отсутствует тело функции"),  ERROR\_ENTRY(604, "Ошибка синтаксического анализа: Неверная конструкция в теле функции"),  ERROR\_ENTRY(605, "Ошибка синтаксического анализа: Ошибка в условном выражении"),  ERROR\_ENTRY(606, "Ошибка синтаксического анализа: Ошибка в вызове функции"),  ERROR\_ENTRY(607, "Ошибка синтаксического анализа: Ошибка в арифметическом выражении"),  ERROR\_ENTRY(608, "Ошибка синтаксического анализа: Ошибка в списке параметров при вызове функции"),  ERROR\_ENTRY(609, "Ошибка синтаксического анализа: Ошибка в арифметическом выражении"),  ERROR\_ENTRY(610, "Ошибка синтаксического анализа: Неверная конструкция в теле цикла или условия"), |

Листинг 4.1 – Перечень сообщений синтаксического анализатора

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

Входными параметрами для синтаксического анализатора являются таблица лексем и таблица идентификаторов, полученные в ходе лексического анализа. По окончании разбора формируется дерево разбора, которое выводится в протокол работы –log:

## **4.8 Принцип обработки ошибок**

Обработка ошибок происходит следующим образом:

1. Синтаксический анализатор перебирает все правила и цепочки правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем.
2. Если невозможно подобрать подходящую цепочку, то генерируется соответствующая ошибка.
3. Все ошибки записываются в общую структуру ошибок.
4. В случае нахождения ошибки, после всей процедуры трассировки в протокол будет выведено диагностическое сообщение.

## **4.9 Контрольный пример**

Пример разбора синтаксическим анализатором исходного кода предоставлен в приложении Е в виде фрагмента трассировки и дерева разбора исходного кода.

# **Глава 5. Разработка семантического анализатора**

## **5.1 Структура семантического анализатора**

Семантический анализ в трансляторе выделен в отдельную фазу, идущую после синтаксического анализа. Он имеет ряд семантических проверок, которые помогают определить логические несоответствия конструкций языка программирования HTA-2021.

## **5.2 Функции семантического анализатора**

За семантический анализ отвечает функция CheckSemantic. На вход она принимает таблицы лексем и идентификаторов, а также поток вывода в протокол.

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

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

Сообщения, формируемые семантическим анализатором, представлены в листинге 5.1.

|  |
| --- |
|  |

Рисунок 5.1 – Перечень сообщений семантического анализатора

## **5.4 Принцип обработки ошибок**

Принцип обработки ошибок семантического анализатора основан на перебирании лексем. Когда встречается лексема, последующая цепочка которой может содержать семантическую ошибку, начинается обращение к таблице идентификаторов. Например, при нахождении лексемы «=», анализатор проверяет последующую цепочку до конца строки на ряд семантических ошибок.

## **5.5 Контрольный пример**

Результат работы контрольного примера расположен в приложении Б, где показан результат лексического анализатора, т.к. представленные таблицы лексем и идентификаторов проходят лексическую и семантическую проверки одновременно.

Таблица 5.3 – Тестирование функций

|  |  |
| --- | --- |
| Исходный код с ошибкой | Генерируемое сообщение об ошибке |
| MAJOR  {  define int a = "adss";  } | Ошибка 312: Ошибка семантического анализа: Несоответсвие типов данных |
| {  define int a = P10;  } | Ошибка 302: Ошибка семантического анализа: Не имеется точки входа в main |
| MAJOR  {  define charline a = "adss"+"sads";  } | Ошибка 317: Ошибка семантического анализа: Неправильное использование операторов в выражении |

# **Глава 6. Преобразование выражений**

## **6.1 Выражения, допускаемые языком**

**Выражения, допускаемые языком**

В языке HTA-2021 допускаются выражения, применимые к целочисленным типам данных. В выражениях поддерживаются арифметические операции, такие как +, -, \*, /, % и (), и вызовы функций как операнды арифметических выражений.

Таблица 6.1 – Приоритет операций в языке HTA-2021

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

## **6.2 Польская запись**

Выражения в языке HTA-2021 преобразовываются к обратной польской записи.

Польская запись -форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. Приоритетность операций приведена в таблице 6.1. Известен следующий принцип построения польской записи:

Алгоритм построения:

– исходная строка: выражение;

– результирующая строка: польская запись;

– стек: пустой;

– результирующая строка: польская запись;

– исходная строка просматривается слева направо;

– операнды переносятся в результирующую строку в порядке их следования;

– операция записывается в стек, если стек пуст или в вершине стека лежит отрывающая скобка;

– операция выталкивает все операции с большим или равным приоритетом в результирующую строку;

– запятая не помещается в стек, если в стеке операции, то все выбираются в строку;

– отрывающая скобка помещается в стек;

– закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются;

– закрывающая скобка выталкивает все до открывающей с и генерирует @ – специальный символ, в которого записывается информация о вызываемой функции, а в поле приоритета для данной лексемы записывается число параметров вызываемой функции;

– по концу разбора исходной строки все операции, оставшиеся в стеке, выталкиваются в результирующую строку.

Таблица 6.2 – Пример преобразования выражения в обратную польскую запись

|  |  |  |
| --- | --- | --- |
| Выражение | Стек | Результат |
| (Y+P5) / P2 \*z |  |  |
| Y+P5)/P2\*z | ( |  |
| +P5)/P2\*z | ( | Y |
| P5)/P2\*z | ( + | Y |
| )/P2\*z | ( + | YP5 |
| /P2\*z |  | YP5+ |
| P2\*z | / | YP5+ |
| \*z | / | YP5+P2 |
| z | \* | YP5+P2/z |
|  | \* | YP5+P2/z |
|  |  | YP5+P2/z\* |

**6.3 Программная реализация обработки выражений**

Программная реализация функции перевода в обратную польскую инверсию содержится в функции startPolish, которая принимает параметром таблицу лексем. Она содержит цикл, который при нахождении символа присваивания (=) или возова процедуры вызывает функцию polishNotation и преобразует последующее выражение до конца строки.

После завершение функции startPolish происходит синхронизация индексов таблицы идентификаторов с таблицей лексем, так как лексемы меняют свое положение.

## **6.4 Контрольный пример**

Пример преобразования выражения к польской записи представлен в таблице 6.2. Преобразование выражений в формат польской записи необходимо для построения более простых алгоритмов их вычисления.

**Глава 7. Генерация кода**

**7.1 Структура генератора кода**

В языке HTA-2021 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 ассемблера, который будет являться результатом работы транслятора. В случае возникновения ошибок генерация кода не будет осуществляться. Структура генератора кода HTA-2021 представлена на рисунке 7.1.



Рисунок 7.1 Структура генератора кода

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

Элементы таблицы идентификаторов расположены в разных сегментах языка ассемблера – .data и .const. Идентификаторы языка HTA-2021 размещены в сегменте данных(.data). Литералы – в сегменте констант (.const). Соответствия между типами данных идентификаторов на языке SIA-2020 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка HTA-2021 и языка Ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке HTA-2021 | Тип идентификатора на языке ассемблера | Пояснение |
| int | SDWORD | Хранит целочисленный тип данных со знаком. |
| charline | BYTE | Каждый символ строки типа charline хранится в поле размером 1 байт. |
| char | BYTE | Символьный литерал типа char хранится в поле размером 1 байт. |

## **7.3 Статическая библиотека**

Статическая библиотека реализована на языке программирования C++. Её реализация находится в проекте StaticLibrary, в свойствах которого был выбран пункт «статическая библиотека .lib». В языке HTA-2021 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++. Объявление функций статической библиотеки генерируется автоматически в коде ассемблера. Объявление функций статической библиотеки генерируется автоматически.

|  |  |
| --- | --- |
| Функция | Описание |
| int findSymb(string str,char ch) | Целочисленная функция.Возращает номер первого вхождения символа ch в строку str ( -1 в случае невхождения). |
| Int destiny(int a,int b) | Целочисленная функция. Генерерирует случайное число в диапазоне [a;b] |
| void OutputInt(int value) | Функция для вывода в стандартный поток значения целочисленного идентификатора/литерала. |
| void OutputStr(char\* line) | Функция для вывода в стандартный поток значения строкового идентификатора/литерала. |
| void ShowChar(char ch) | Функция для вывода в стандартный поток значения символьного идентификатора/литерала |

* 1. **Особенности алгоритма генерации кода**

В языке HTA-2021 генерация кода строится на основе таблиц лексем и идентификаторов. Общая схема работы генератора кода представлена на рисунке 7.2

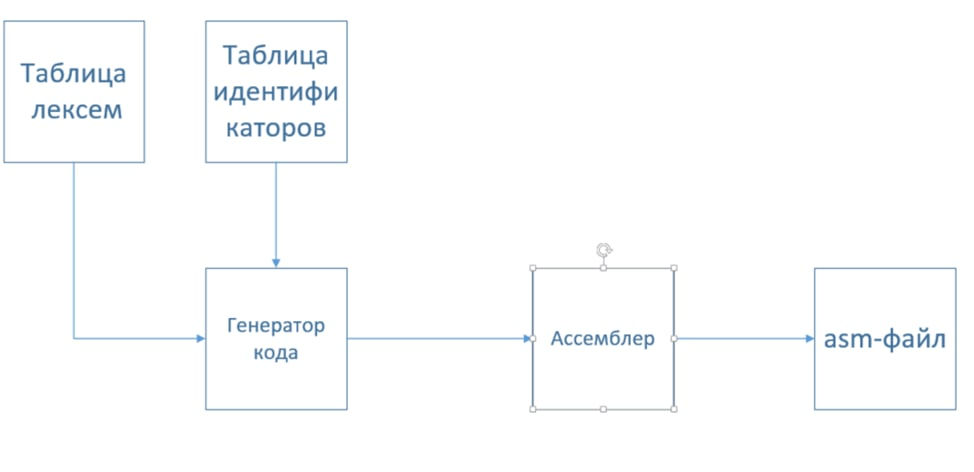


Рисунок 7.2

* 1. **Входные параметры генератора кода**

На вход генератору кода поступают таблицы лексем и идентификаторов исходного код программы на языке KDV-2019. Результаты работы генератора кода выводятся в файл с расширением .asm.

**7.6 Контрольный пример**

Результат генерации ассемблерного кода на основе контрольного примера из приложения А приведен в приложении Д. Результат работы контрольного примера приведён на рисунке 7.3.



Рисунок 7.3 Результат работы программы на языке HTA-2021

**Глава 8. Тестирование транслятора**

## **8.1 Тестирование фазы проверки на допустимость символов**

В данной главе описаны возможные ошибки, возникающие на различных этапах работы транслятора. Результат тестирования представлен в таблице 8.1.

Таблица 8.1 - Тестирование транслятора

Таблица 8.1 – Тестирование фазы проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| int Minor ك() {ret 0;} | Ошибка 200: [ LEXICAL ] Недопустимый символ в исходном файле (-in) |

## **8.2 Тестирование лексического анализатора**

На этапе лексического анализа могут возникнуть ошибки, описанные в пункте 3.7. Результаты тестирования лексического анализатора показаны в таблице 8.2.

Таблица 8.2 – Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| MAJOR  {  string ?as ;  } | Ошибка 205: [ LEXICAL ] Неизвестная последовательность символов |

## **8.3 Тестирование синтаксического анализатора**

На этапе синтаксического анализа могут возникнуть ошибки, описанные в пункте 4.6. Результаты тестирования синтаксического анализатора показаны в таблице 8.3.

Таблица 8.3 – Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| int function fi(int a, int b)  {  ret 0;  }  main  {  let int i = fi(1, );  } | Cтрока 8, [ SYNTAX ] Ошибка в списке параметров при вызове функции |

## **8.4 Тестирование семантического анализатора**

Итоги тестирования семантического анализатора приведены в пункте 5.5

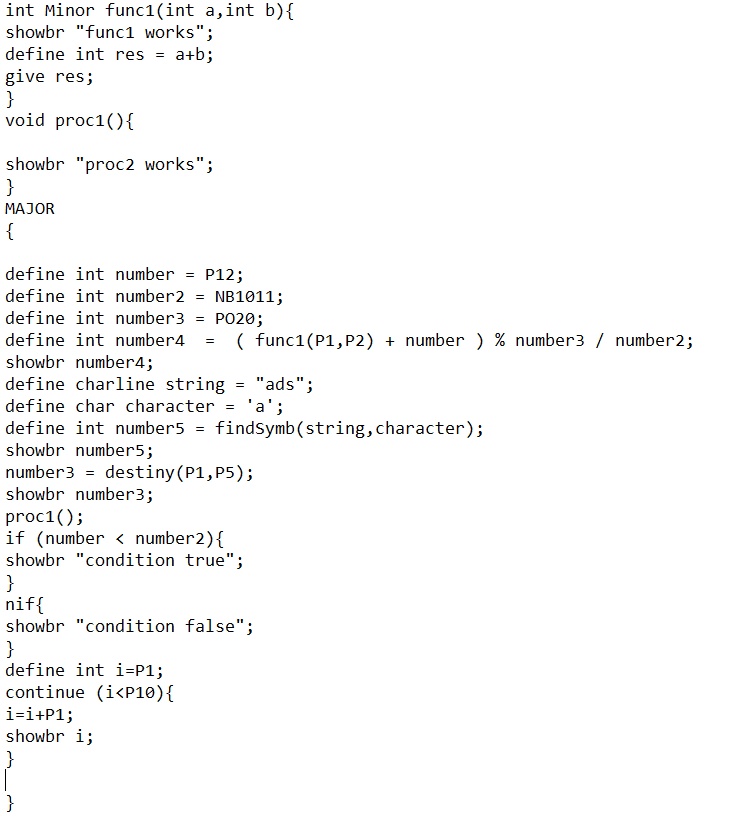
# **Заключение**

В ходе выполнения курсовой работы был разработан транслятор для языка программирования HTA-2021. Таким образом, были выполнены основные задачи данной курсовой работы:   
- Сформулирована спецификация языка HTA-2021;   
- Разработаны конечные автоматы и алгоритмы для реализация лексического анализатора;   
- Разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;   
- Разработан семантический анализатор, осуществляющий проверку смысла используемых инструкций;   
- Разработан транслятор с языка программирования HTA-2021 на язык низкого уровня Assembler;   
- Проведено тестирование всех вышеперечисленных компонентов.   
- Реализованы 5 арифметических оператора для вычисления выражений, два типа данных, функции;   
- присутствует подключаемая стандартная библиотека;   
- обрабатывается 30 возможных исключительных ситуаций;   
- поддерживается оператор вывода;   
- среднее время выполнения генерации кода - 0.2 секунды;   
- количество строк – около 4000.

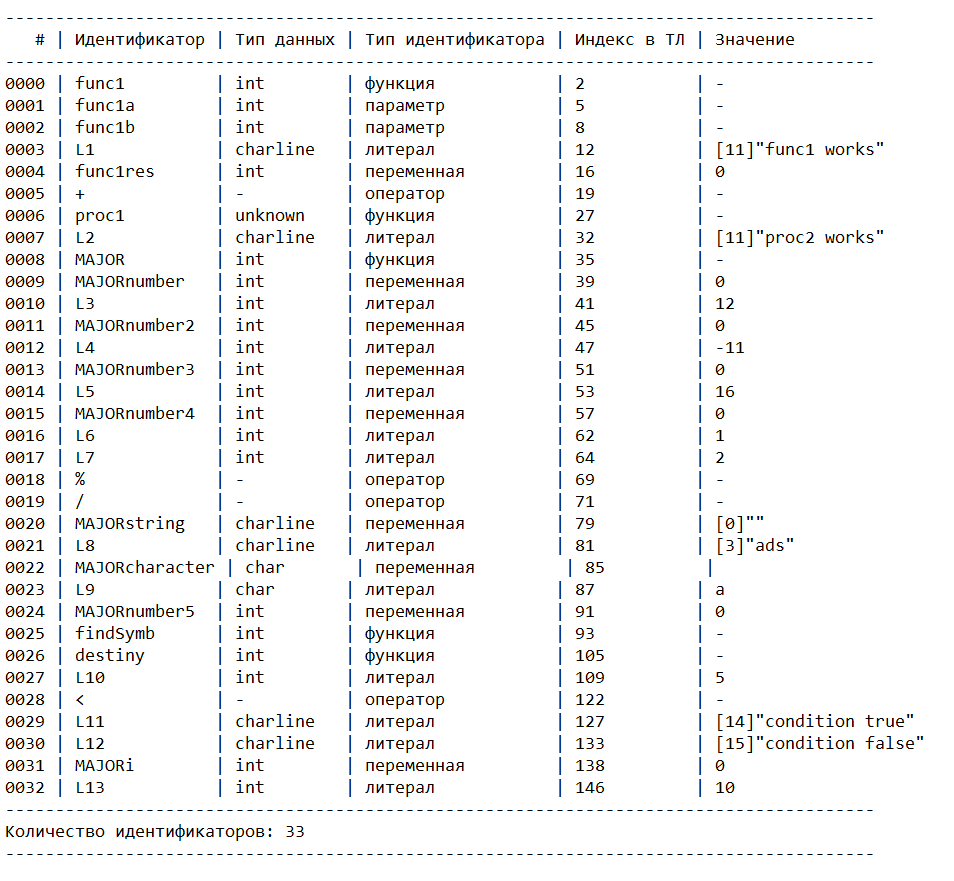
# **Список использованной литературы**

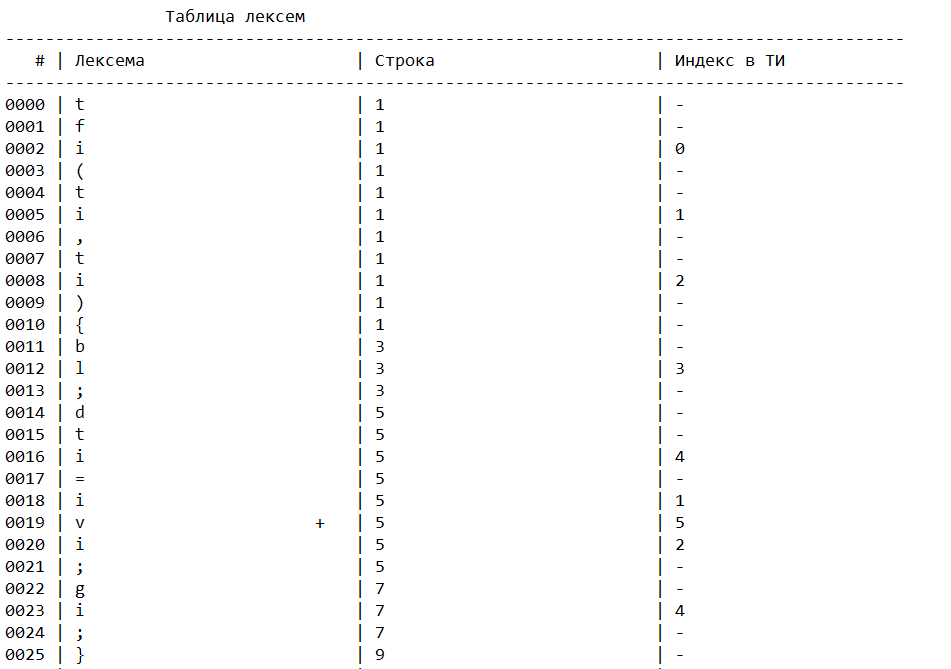
1. Ахо А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003. – 768с.
2. Ахо, А. Теория синтаксического анализа, перевода и компиляции /А. Ахо, Дж. Ульман. – Москва : Мир, 1998. – Т. 2 : Компиляция. - 487 с.
3. Ирвин К. Р. Язык ассемблера для процессоров Intel / К. Р. Ирвин. – M.: Вильямс, 2005. – 912с.
4. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. - 3-е изд. – Москва : Вильямс, 2003. - 429 с.

# **Приложение А**

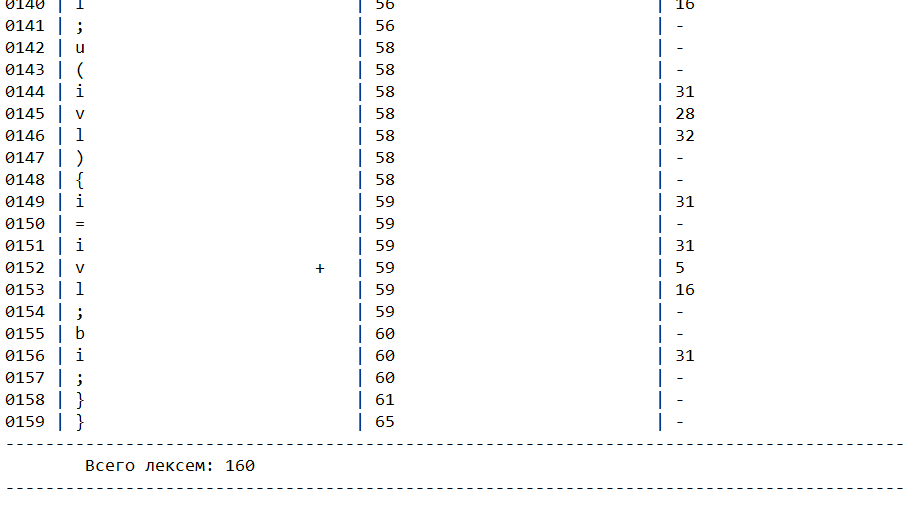
****

# **Приложение Б**

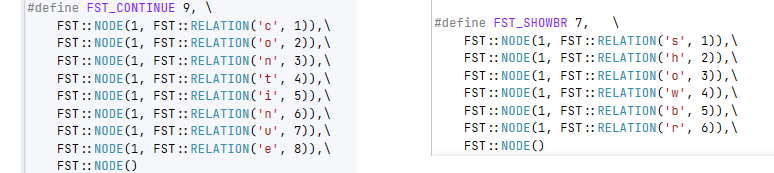
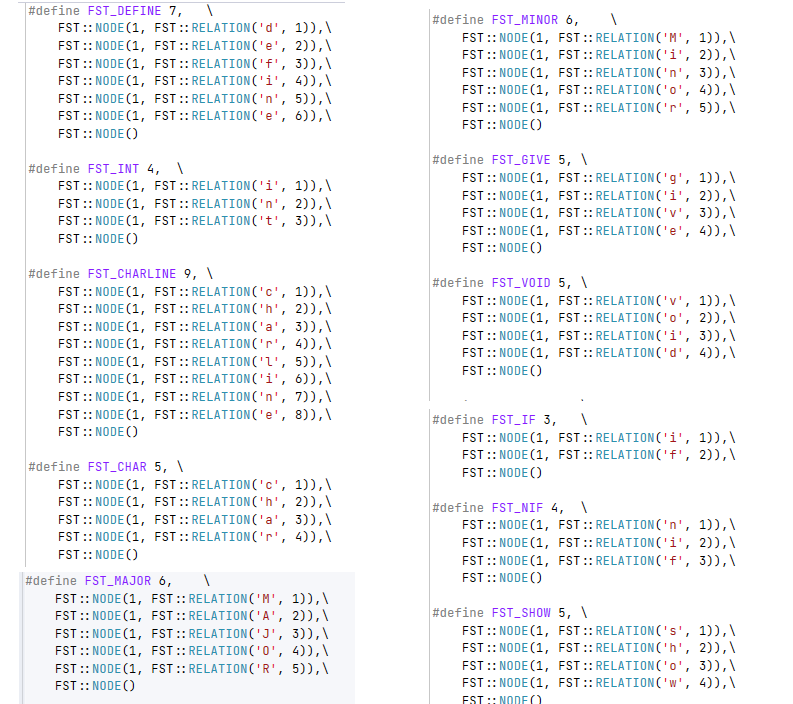
****

Начало таблицы лексем: ****

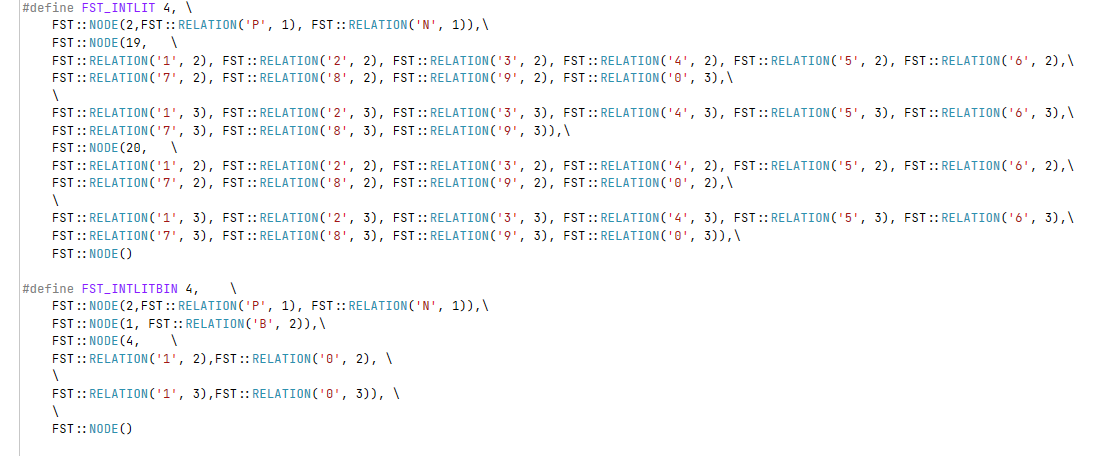
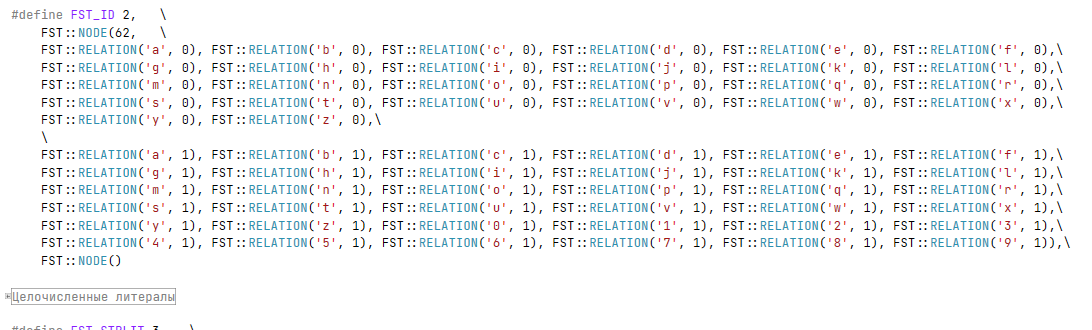
Окончание таблицы лексем:

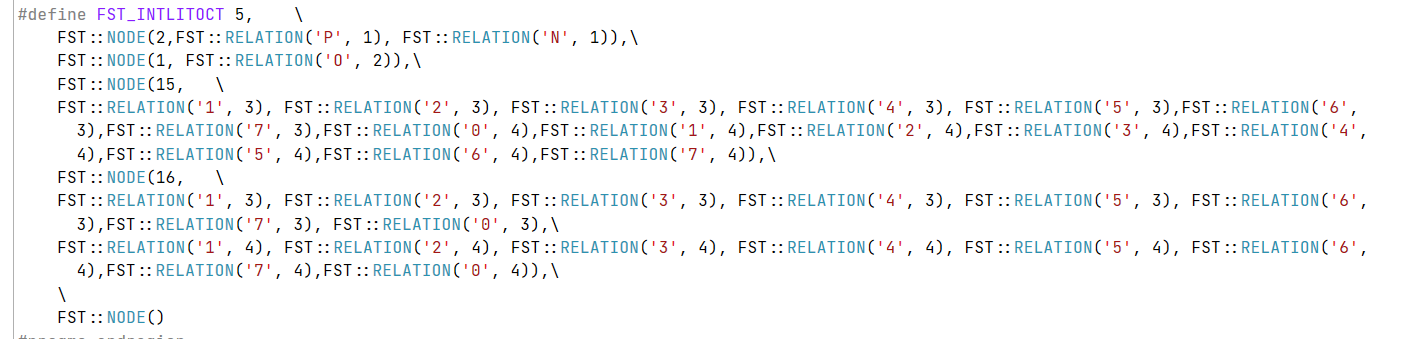


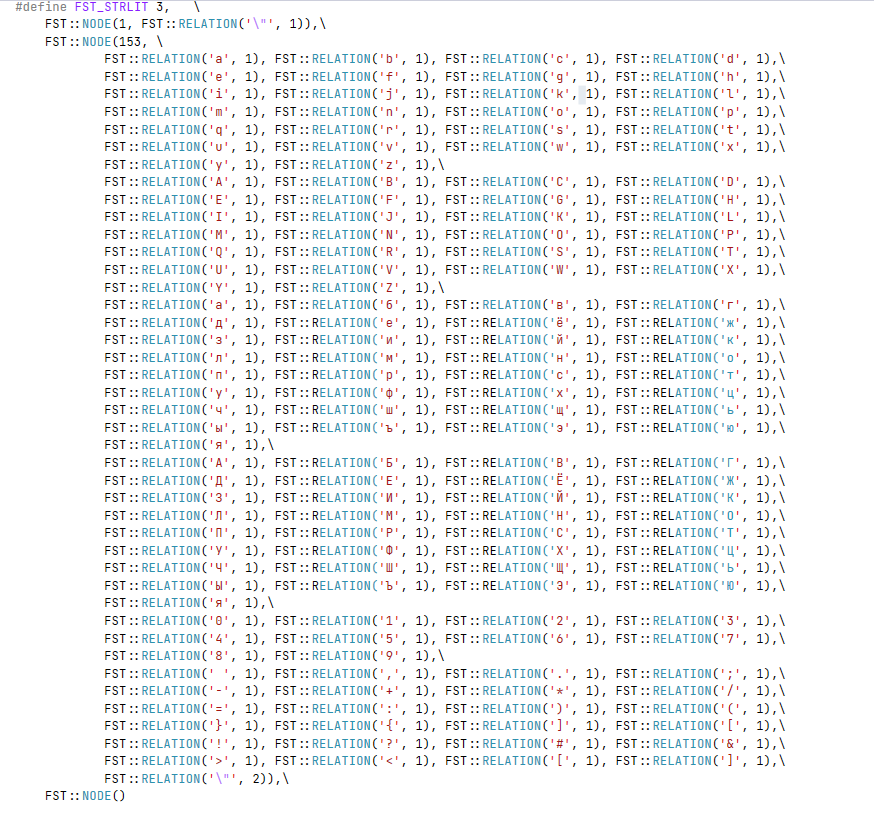
# **Приложение В**

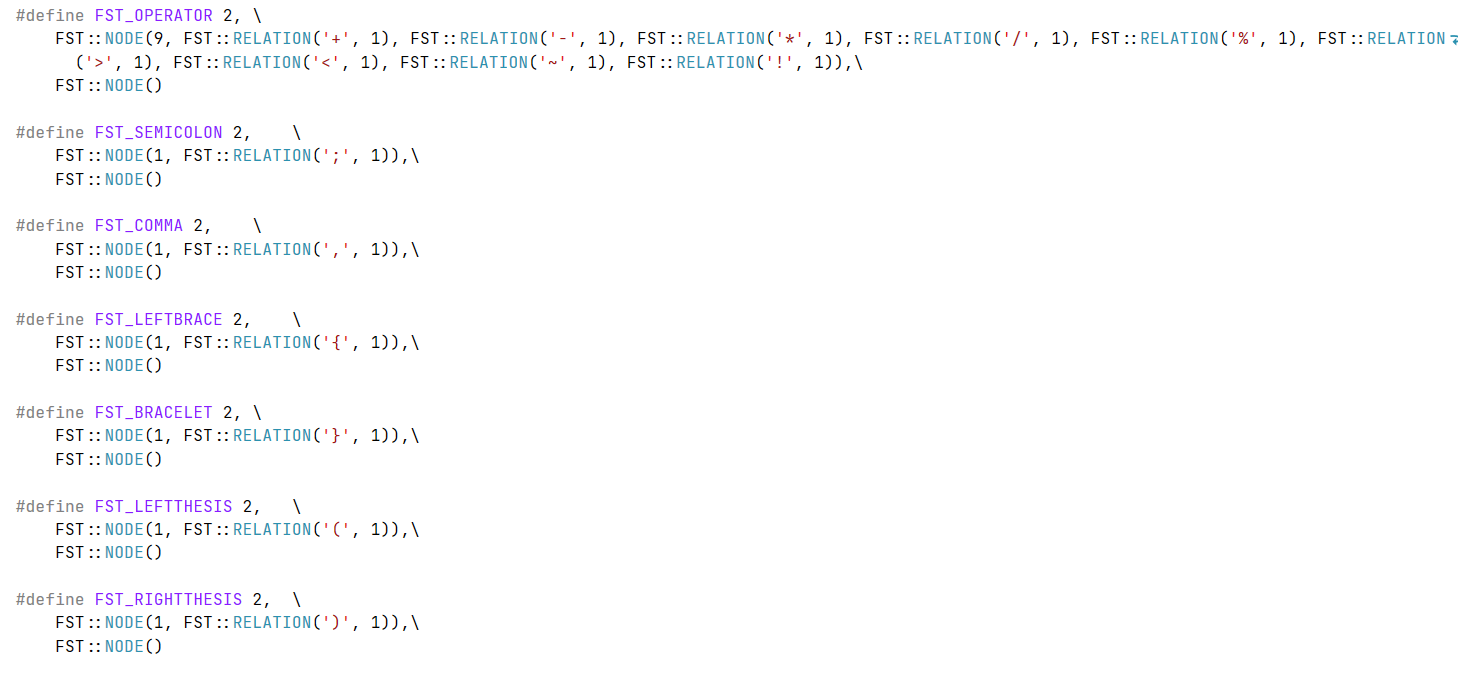


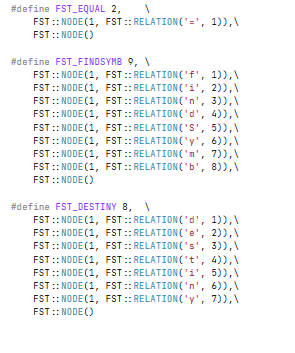
# 



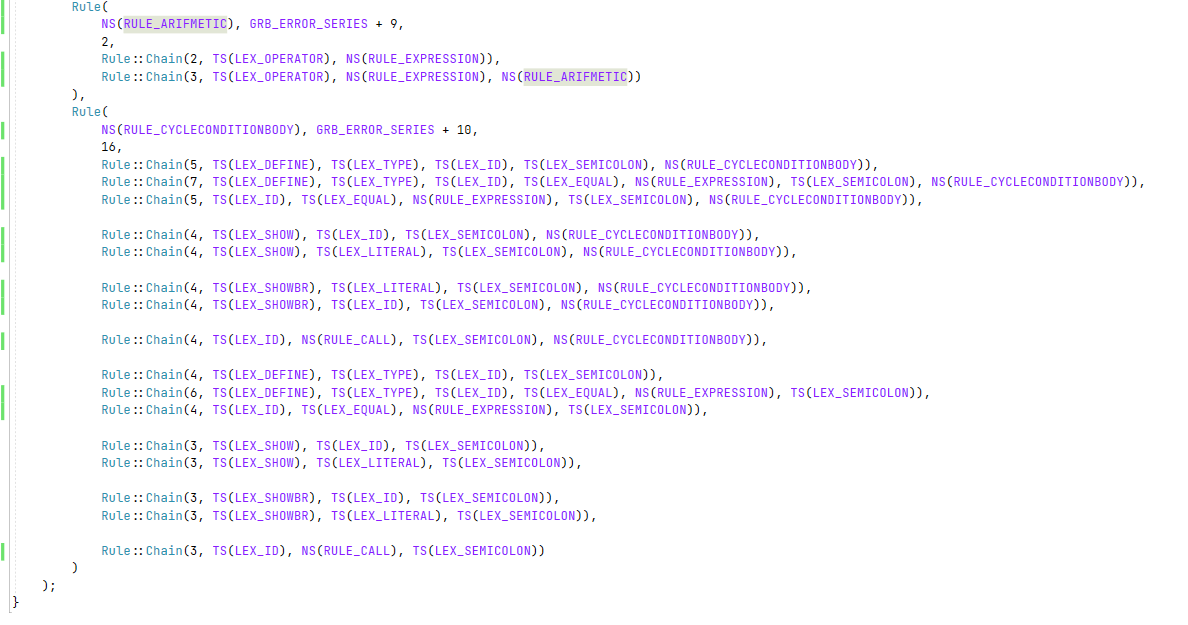
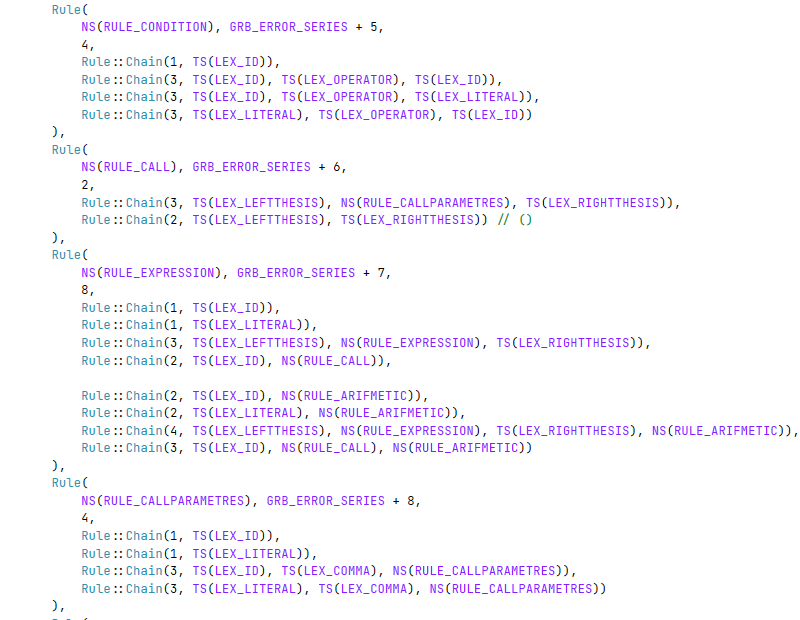
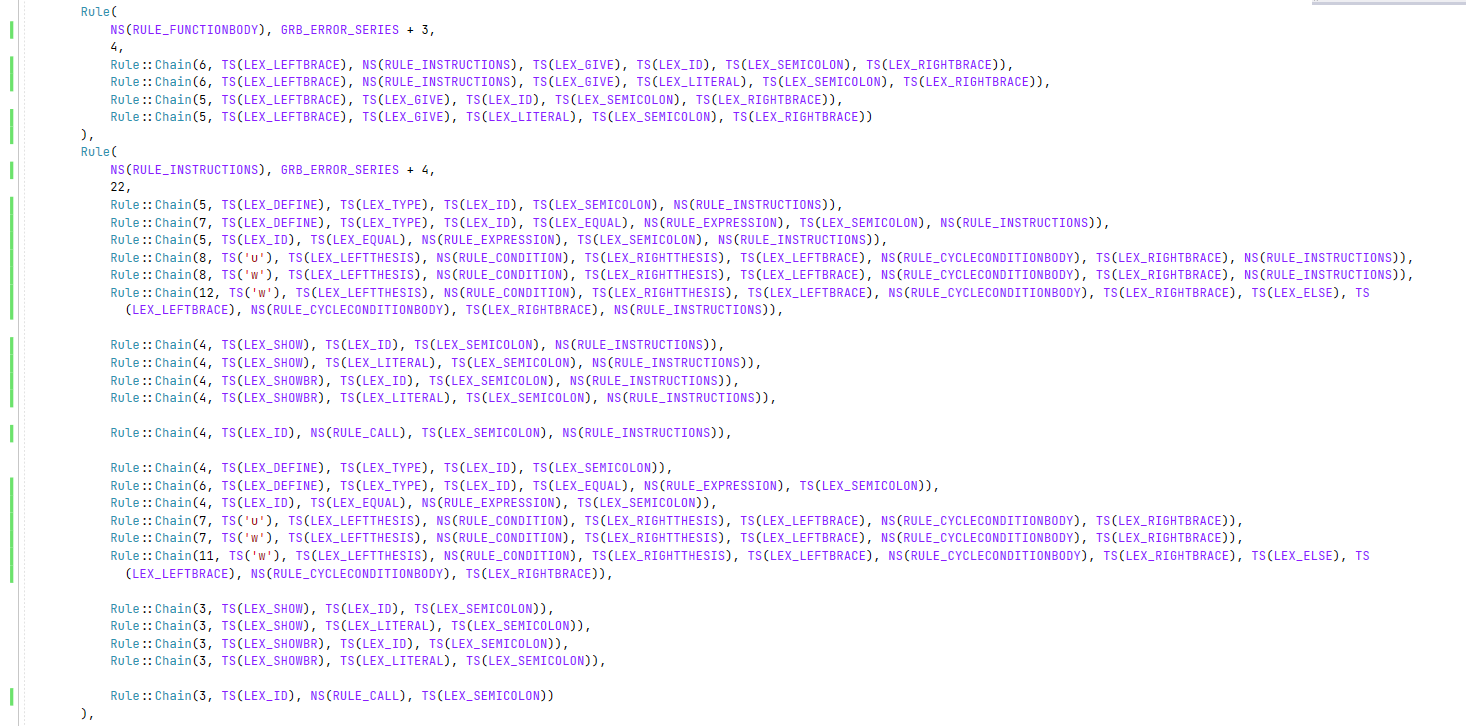
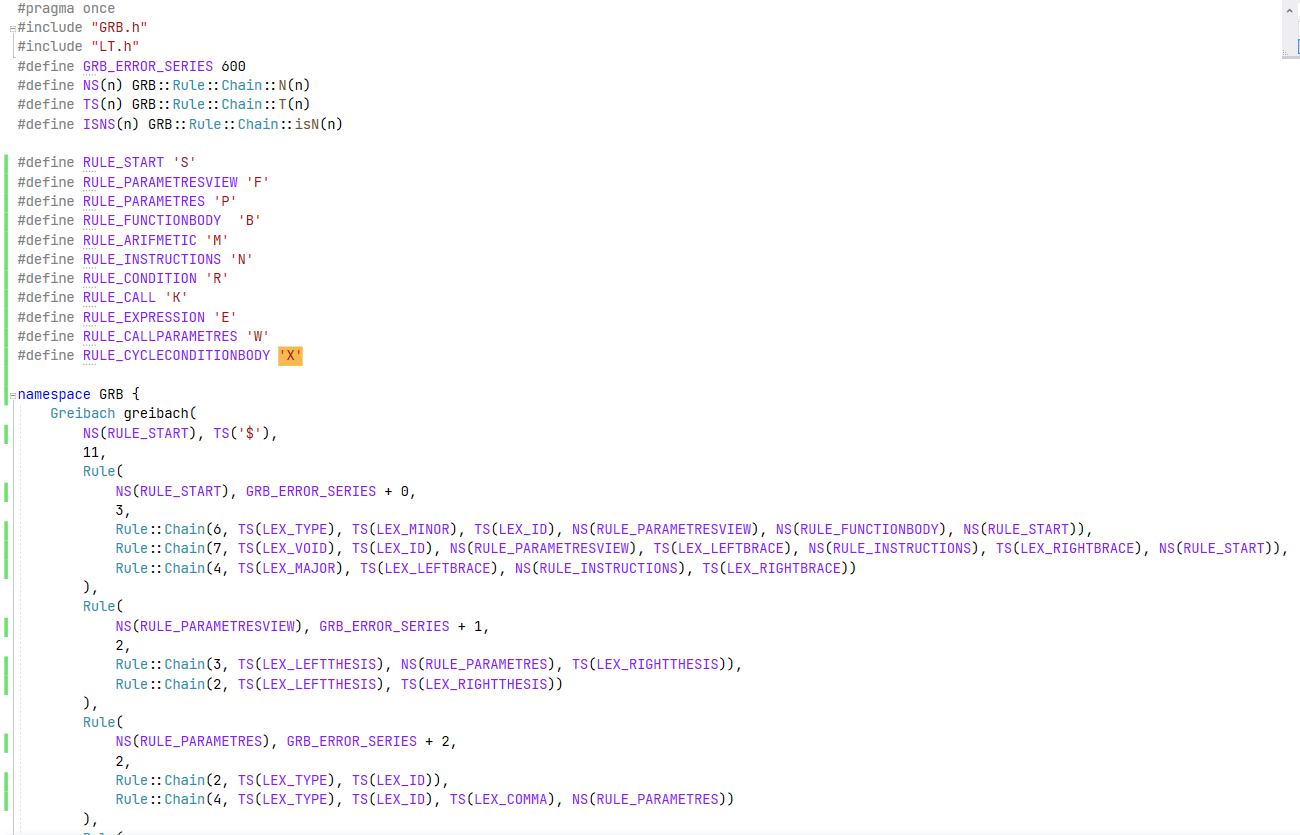




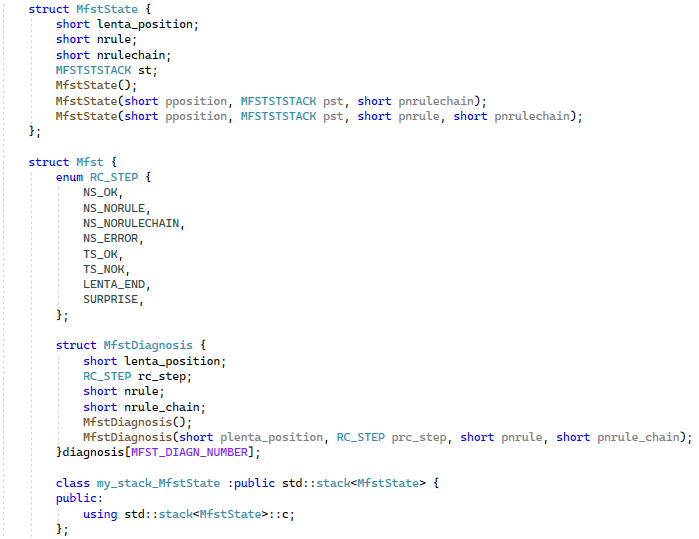
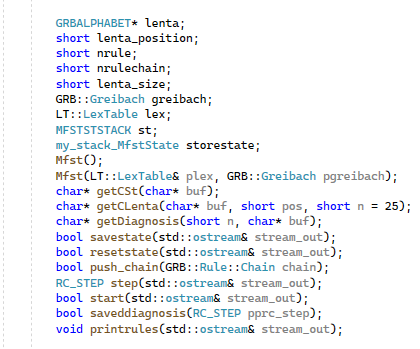


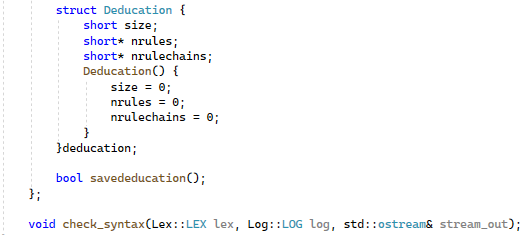


# **Приложение Г**



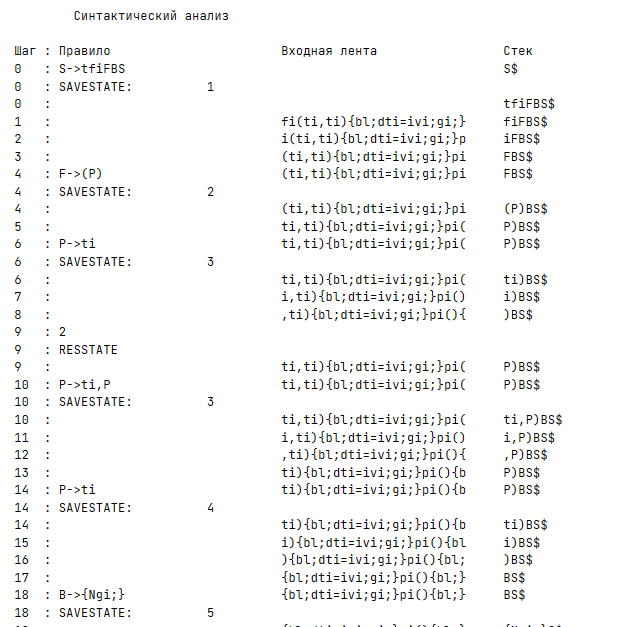
# **Приложение Д**



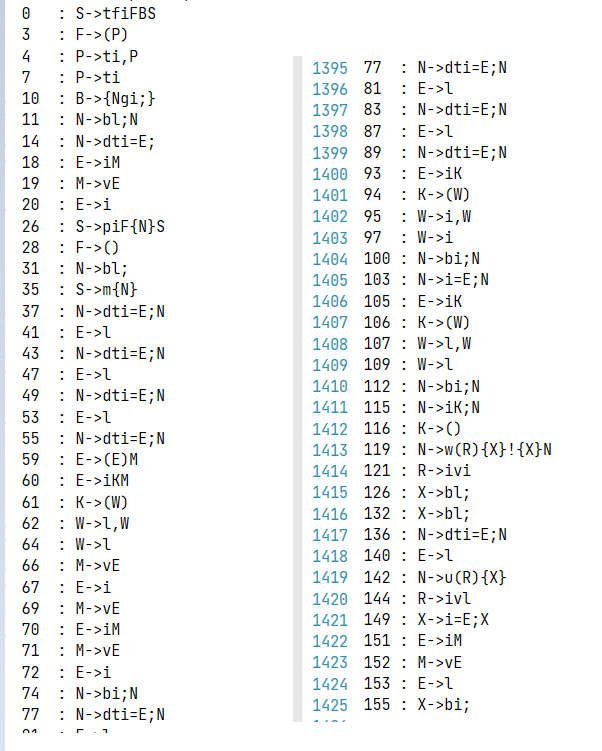
# **Приложение Е**

Начало разбора

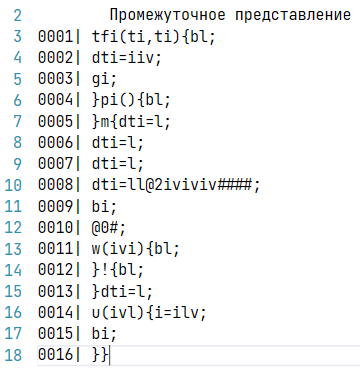


Конец разбора

Дерево разбора



# **Приложение Ж**



# **Приложение З**

# 

