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 GAA-2020»

Выполнил студент Губарь Анна Александровна

(Ф.И.О.)

Руководитель проекта преп.-стаж. Котович Д. В.

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

Заведующий кафедрой к.т.н., доц. Пацей Н.В.

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

Консультанты преп.-стаж. Котович Д. В.

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

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

Нормоконтролер преп.-стаж. Котович Д. В.

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

Курсовой проект защищен с оценкой

Минск 2020

**Содержание**

[**Введение:** 4](#_Toc59020686)

[**Глава 1. Спецификация языка программирования** 5](#_Toc59020687)

[**1.1 Характеристика языка программирования** 5](#_Toc59020688)

[**1.2 Определение алфавита языка программирования** 5](#_Toc59020689)

[**1.3 Применяемые сепараторы** 5](#_Toc59020690)

[**1.4 Применяемые кодировки** 6](#_Toc59020691)

[**1.5 Типы данных** 6](#_Toc59020692)

[**1.6 Преобразование типов данных** 7](#_Toc59020693)

[**1.7 Идентификаторы** 7](#_Toc59020694)

[**1.8 Литералы** 7](#_Toc59020695)

[**1.9 Объявление данных** 8](#_Toc59020696)

[**1.10 Инициализация данных** 9](#_Toc59020697)

[**1.11 Инструкции языка** 9](#_Toc59020698)

[**1.12 Операции языка** 9](#_Toc59020699)

[**1.13** **Выражения и их вычисления** 10](#_Toc59020700)

[**1.14 Конструкции языка** 10](#_Toc59020701)

[**1.15 Область видимости идентификаторов** 11](#_Toc59020702)

[**1.16 Семантические проверки** 11](#_Toc59020703)

[**1.17 Распределение оперативной памяти на этапе выполнения.** 12](#_Toc59020704)

[**1.18 Стандартная библиотека и ее состав** 12](#_Toc59020705)

[**1.19 Ввод и вывод данных** 12](#_Toc59020706)

[**1.20 Точка входа** 12](#_Toc59020707)

[**1.21 Препроцессор** 12](#_Toc59020708)

[**1.22 Соглашения о вызове** 13](#_Toc59020709)

[**1.23 Объектный код** 13](#_Toc59020710)

[**1.24 Классификация сообщений транслятора** 13](#_Toc59020711)

[**1.25 Контрольный пример** 13](#_Toc59020712)

[**Глава 2. Структура транслятора** 13](#_Toc59020713)

[**2.1 Компоненты транслятора, их назначение и принципы взаимодействия** 14](#_Toc59020714)

[**2.2 Перечень входных параметров транслятора** 15](#_Toc59020715)

[**2.3 Перечень протоколов, формируемых транслятором и их содержимое** 15](#_Toc59020716)

[**Глава 3. Разработка лексического анализатора** 16](#_Toc59020717)

[**3.1 Структура лексического анализатора** 16](#_Toc59020718)

[**3.2 Контроль входных символов** 16](#_Toc59020719)

[**3.3 Удаление избыточных символов** 18](#_Toc59020720)

[**3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем** 18](#_Toc59020721)

[**3.5 Основные структуры данных** 19](#_Toc59020722)

[**3.6 Структура и перечень сообщений лексического анализатора** 19](#_Toc59020723)

[**3.7 Принцип обработки ошибок** 19](#_Toc59020724)

[**3.8 Параметры лексического анализатора** 19](#_Toc59020725)

[**3.9 Алгоритм лексического анализа** 19](#_Toc59020726)

[**3.10 Контрольный пример** 19](#_Toc59020727)

[**Глава 4. Разработка синтаксического анализатора** 20](#_Toc59020728)

[**4.1 Структура синтаксического анализатора** 20](#_Toc59020729)

[**4.2 Контекстно-свободная грамматика, описывающая синтаксис языка** 20](#_Toc59020730)

[**4.3 Построение конечного магазинного автомата** 21](#_Toc59020731)

[**4.4 Основные структуры данных** 22](#_Toc59020732)

[**4.5 Описание алгоритма синтаксического разбора** 22](#_Toc59020733)

[**4.6 Структура и перечень сообщений синтаксического анализатора** 22](#_Toc59020734)

[**4.7 Параметры синтаксического анализатора и режимы его работы** 22](#_Toc59020735)

[**4.8 Принцип обработки ошибок** 23](#_Toc59020736)

[**4.9 Контрольный пример** 23](#_Toc59020737)

[**Глава 5. Разработка семантического анализатора** 23](#_Toc59020738)

[**5.1 Структура семантического анализатора** 23](#_Toc59020739)

[**5.2 Функции семантического анализатора** 23](#_Toc59020740)

[**5.3 Структура и перечень сообщений семантического анализатора** 24](#_Toc59020741)

[**5.4 Принцип обработки ошибок** 24](#_Toc59020742)

[**5.5 Контрольный пример** 24](#_Toc59020743)

[**Глава 6. Преобразование выражений** 25](#_Toc59020744)

[**6.1. Выражения, допускаемые языком** 25](#_Toc59020745)

[**6.2 Польская запись и принцип ее построения** 25](#_Toc59020746)

[**6.3 Программная реализация обработки выражений** 26](#_Toc59020747)

[**6.4 Контрольный пример** 26](#_Toc59020748)

[**Глава 7. Генерация кода** 27](#_Toc59020749)

[**7.1 Структура генератора кода** 27](#_Toc59020750)

[**7.2 Представление типов данных в оперативной памяти** 27](#_Toc59020751)

[**7.3 Статическая библиотека** 27](#_Toc59020752)

[**7.4 Входные параметры генератора кода** 28](#_Toc59020753)

[**7.5 Контрольный пример** 28](#_Toc59020754)

[**Глава 8. Тестирование транслятора** 29](#_Toc59020755)

[**8.1 Тестирование лексического анализатора** 29](#_Toc59020756)

[**8.3 Тестирование синтаксического анализатора** 29](#_Toc59020757)

[**8.3 Тестирование семантического анализатора** 29](#_Toc59020758)

[**Заключение** 30](#_Toc59020759)

[**Приложение А** 31](#_Toc59020760)

[**Приложение Б** 32](#_Toc59020761)

[**Приложение В** 33](#_Toc59020762)

[**Приложение Г** 34](#_Toc59020763)

[**Приложение Д** 36](#_Toc59020764)

[**Приложение Е** 38](#_Toc59020765)

[**Приложение Ж** 39](#_Toc59020766)

[**Приложение З** 43](#_Toc59020767)

[**Приложение И** 45](#_Toc59020768)

# **Введение:**

Основной целью данной курсовой работы является разработка транслятора для языка программирования GAA-2020. Главная задача транслятора заключается в том, чтобы исходный код на языке GAA-2020 в код на языке Assembler.

Исходя из цели курсового проекта, были определены следующие задачи:

– разбработка спецификации языка программирования;

– разбратка структуры транслятора;

– разработка лексического анализатора;

– разработка синтаксического анализатора;

– разработка семантического анализатора;

– обработка выражений;

– генерация кода на язык Assebler;

– тестирование транслятора.

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

* спецификация языка программирования;
* структура транслятора;
* лексический анализатор;
* синтаксический анализатор;
* семантический анализатор;
* преобразование выражений;
* генерация кода;
* тестирование транслятора.

Язык программирования GAA-2020 предназначен для работы с консолью, выполнения простейших арифметический действий и операций над строками.

# **Глава 1. Спецификация языка программирования**

# **1.1 Характеристика языка программирования**

GAA-2020 является компилируемым процедурным языком высокого уровня.

Поддерживает 2 типа данных: строковый (string) и целочисленный (integer). В стандартной библиотеке языка содержится 2 функции для работы со строками: copystr (копирование строк) и joinstr (конкатенация строк). Преобразование типов в языке GAA-2020 не поддерживается.

# **1.2 Определение алфавита языка программирования**

Для записи инструкций языка используются символы: [a…z], [A…Z].

Для записи литералов (строковых и целочисленных) используют символы: [a…я], [a…z], [0…9].

В качестве сепараторов и специальных символов используются: () {} ~ + - \* / % = , #; «пробел»

# **1.3 Применяемые сепараторы**

Символы-сепараторы необходимы для эффективного разбиения исходного текста программы на лексемы на этапе лексического анализа.

Таблица 1.1 Символы-сепараторы

|  |  |
| --- | --- |
| Символы | Назначение |
| (…) | Повышение приоритета арифметических операций, ограничение блока параметров функции, условие окончания цикла. |
| {…} | Ограничение тела функции или цикла. |
| ~ | Разделитель программных конструкций. |
| + - \* / % | Арифметические операции. |
| = | Оператор присваивания. |
| , | Разделитель параметров функции. |
| # | Начало строки комментария. |
| ; | Конец строки комментария. |
| «пробел» | Разделитель цепочек языка, не допустим и именах идентификаторов и ключевых словах языка. |

# **1.4 Применяемые кодировки**

Для написания программ на языке GAA-2020 применяется кодировка Windows-1251.

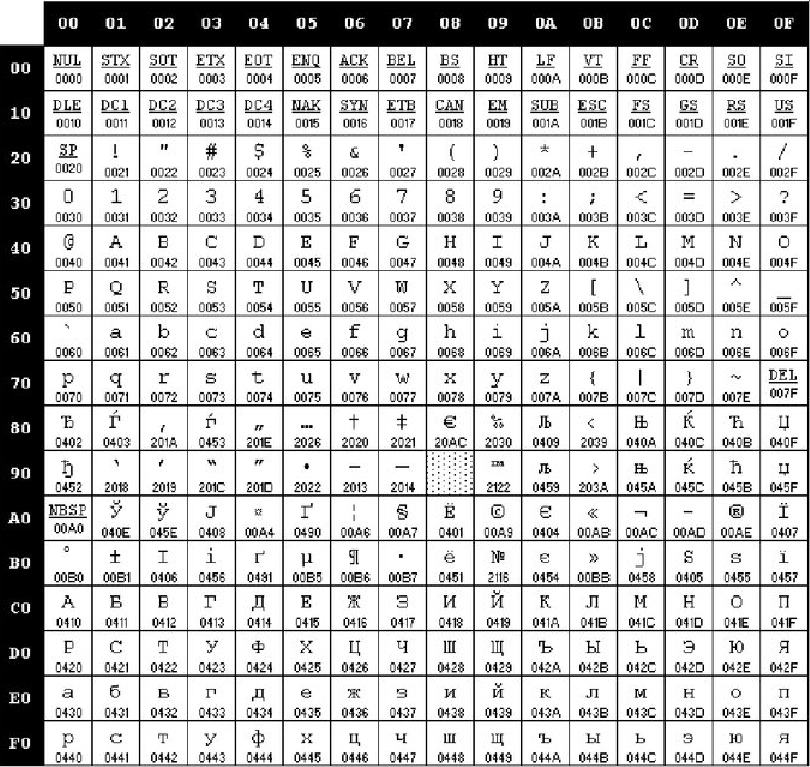


Рисунок 1.1 Алфавит входных символов

# **1.5 Типы данных**

В языке GAA-2020 реализованы 2 фундаментальных типа данных: целочисленный и строковый.

Таблица 1.2 Характеристика типов данных языка GAA-2020

|  |  |
| --- | --- |
| Тип данных | Характеристика |
| Строковый  (string) | По умолчанию инициализируется пустой строкой: ‘’.  Размер в байтах (для одного символа): 1 байт.  Допустимый диапазоны значений: от 0 до 255.  Применяемые операции:  + (бинарный) – конкатенация. |

Продолжение таблицы 1.2 Характеристика типов данных языка GAA-2020

|  |  |
| --- | --- |
| Тип данных | Характеристика |
| Целочисленный  (integer) | По умолчанию инициализируется 0.  Размер в байтах: 4.  Допустимый диапазоны значений: от −2147483648 до 2147483647.  Применяемые операции:  + (бинарный) – сложение;  - (бинарный) – вычитание;  - (унарный) – отрицательное число;  \* (бинарный) – умножение;  / (бинарный) – деление;  % (бинарный) – остаток от деления; |

# **1.6 Преобразование типов данных**

Преобразование типов данных в языке GAA-2020 не поддерживается.

# **1.7 Идентификаторы**

Общее число идентификаторов не должно превышать максимальный размер таблицы идентификаторов (4096). Длина идентификатора не должна превышать 7 символов. В именах идентификаторов допустимо использование символов верхнего и нижнего регистров. Идентификаторы не должны совпадать с ключевыми словами языка. Кии типам идентификаторов относятся: имя переменной, имя функции, имя параметра.

# **1.8 Литералы**

В языке GAA-2020 предусмотрены строковые и целочисленные литералы. Язык поддерживает представление целочисленных литералов в десятичной, восьмеричной и двоичной системе счисления.

Таблица 1.3 Характеристика литералов языка GAA-2020

|  |  |
| --- | --- |
| Литералы | Характеристика |
| Строковые | Набор символов алфавита языка, заключенный в одинарные кавычки (‘’).  Допустимо использование символов только нижнего регистра. |

Продолжение таблицы 1.3 Характеристика литералов языка GAA-2020

|  |  |
| --- | --- |
| Литералы | Характеристика |
| Целочисленные в двоичной форме | Последовательность цифр 0 и 1 с обязательным префиксом “Bx”: Bx101. |
| Целочисленные в восьмеричной форме | Последовательность цифр 0…7 с обязательным префиксом “Ox”: Ox172. |
| Целочисленные в десятичной форме | Последовательность цифр 0…9 без префиксов: 79. |
| Отрицательные целочисленные | Для того, чтобы получить отрицательное значение литерала, необходимо использовать унарный оператор “-“: -Ox172, -8. |

# **1.9 Объявление данных**

Для объявления переменной используется ключевое слово **new**, после которого необходимо указать тип данных и имя идентификатора.

Объявление без инициализации:

**new integer** x~

Объявление с инициализацией:

**new string** m=’hello’~

Для объявления функции необходимо использовать ключевое слово **function**, предварительно указав тип функции. Далее необходимо указать имя функции, список параметров и ее тело. Для объявления параметра необходимо перед именем идентификатора указать тип данных:

**integer function** Func(**integer** a, **integer** b)

{

…

}~

# **1.10 Инициализация данных**

По умолчанию переменные типа integer инициализируются нулём, а переменные типа string – пустой строкой (‘’). В языке GAA-2020 также допустима инициализация при объявлении. При этом переменно присваивается значение литерала или идентификатора, находящееся справа от знака равенства.

# **1.11 Инструкции языка**

Таблица 1.4 Характеристика инструкций языка GAA-2020

|  |  |
| --- | --- |
| Инструкция | Описание |
| Объявление переменной | **new** <тип данных><идентификатор>~  Пример: **new string** q~ |
| Возврат значения функции | **return** <идентификатор/литерал>~  Пример: **return** m~ **return** 4~ |
| Вывод данных | **print** <идентификатор/литерал>~  Пример: **print** 'hi'~ **print** Func(4,3)~ |
| Вызов функции | <идентификатор>(<список параметров>)~  Пример: fi(z)~ fi(1)~ |
| Операция присваивания | <идентификатор>=<литерал/идентификатор>~  Пример: g=joinst(a,b)~ |

# **1.12 Операции языка**

Таблица 1.5 Характеристика операций языка GAA-2020

|  |  |  |
| --- | --- | --- |
| Оператор | Характеристика | Приоритет |
| + (для данных типа integer) | сложение | 2 |
| + (для данных типа string) | конкатенация | 2 |
| - (бинарный) | вычитание | 2 |
| - (унарный) | отрицательное значение целочисленного литерала | - |
| \* | умножение | 3 |
| / | деление | 3 |
| % | остаток от деления | 3 |
| () | повышение приоритета операций | 4 |

# **1.13** **Выражения и их вычисления**

В языке GAA-2020 допустимы выражения как с целочисленными, так и со строковыми литералами. При этом в выражениях со строковыми операндами разрешается использовать только оператор “+”, помимо этого такое выражение не должно содержать более двух операндов.

Выражения языка GAA-2020 подчиняются следующим правилам:

* выражение, как и любая инструкция языка, должна заканчиваться сепаратором “~”;
* использование в одном выражении операндов разных типов недопустимо;
* присваивать идентификатору результат выражения разрешено только если их типы данных совпадают;
* разрешено изменять приоритет операций, используя оператор “()”;
* недопустимо использование двух арифметических операторов подряд;
* разрешено использовать в качестве операнда вызов функции;

# **1.14 Конструкции языка**

Таблица 1.6 Характеристика конструкций языка GAA-2020

|  |  |
| --- | --- |
| Конструкция | Характеристика |
| Главная функция | **head**  {  …  **return** 0~  }~ |
| Функция | <тип данных> **function** <идентификатор> (<тип данных> <идентификатор>, …)  {  …  **return** <идентификатор/литерал>~  }~  Функции пользователя должны быть определены вне тела главной функции. |
| Цикл | **until**(<идентификатор>=<литерал>~)  {  …  }~  Инструкции в теле цикла повторяются пока идентификатор не окажется равен указанному значению. |
| Строка комментария | #<литерал>; |

# **1.15 Область видимости идентификаторов**

Все переменные и параметры локальны и могут объявляться только в теле функции. Идентификаторы, объявленные в одной функции, недоступны любой другой.

# **1.16 Семантические проверки**

1. Проверка наличия главной функции head – точки входа в программу.
2. Проверка количества точек входа в программу (не должно превышать 1).
3. Проверка на наличие повторного объявления идентификатора.
4. Проверка соответствия формальных и фактических параметров функции по типу и количеству.
5. Проверка совпадения типа функции и типа возвращаемого ею значения.
6. Проверка на использование идентификатора до его объявления.
7. Проверка соответствия типа идентификатора и типа присваиваемого ему значения.
8. Проверка использования однотипных операндов в выражениях.
9. Проверка значений целочисленных литералов на допустимость (входит ли в разрешенный диапазон).
10. Проверка операций со строками на допустимость.

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

При трансляции кода используется две области памяти: сегмент констант и сегмент данных. Они заполняются с помощью таблицы лексем и таблицы идентификаторов, сформированных на этапе лексического анализа. В сегмент констант заносятся строковые и целочисленные литералы. В сегмент данных – переменные и параметры функций.

# **1.18 Стандартная библиотека и ее состав**

Таблица 1.7 Стандартная библиотека языка GAA-2020

|  |  |
| --- | --- |
| Функция | Характеристика |
| string joinst(string a, string b) | Строковая функция, возвращающая результат конкатенации строк а и b, записанный в строку a |
| copystr(string a, string b) | Строковая функция, возвращающая строку b, скопированную в строку a. |

# **1.19 Ввод и вывод данных**

Вывод данных осуществляется с помощью ключевого слова **print**. В качестве аргумента принимаются литералы, идентификаторы и выражения:

print 4~

print z~

print m+1~

Вывод данных не предусмотрен.

# **1.20 Точка входа**

Точкой входа является функция head. Точка входа в приложение не может отсутствовать, а также не может быть переопределена.

# **1.21 Препроцессор**

В языке GAA-2020 не предусмотрена реализация препроцессора.

# **1.22 Соглашения о вызове**

При генерации кода используется соглашение о вызове **stdcall**. Параметры функций передаются через стек справа налево. Память освобождает вызываемый код.

# **1.23 Объектный код**

Исходный код языка транслируется в язык ассемблера.

# **1.24 Классификация сообщений транслятора**

Определено 4 типа ошибок: ошибки лексического анализа, ошибки синтаксического анализа, ошибки семантического анализа, критические системные ошибки.

Таблица 1.8 Префиксы сообщений об ошибках

|  |  |
| --- | --- |
| Префикс | Описание |
| Syn | Найдена синтаксическая ошибка. |
| Sem | Найдена семантическая ошибка. |
| Lex | Найдена лексическая ошибка. |

Отсутствие префикса говорит о том, что найдена критическая системная ошибка.

# **1.25 Контрольный пример**

Контрольный пример представлен в приложении А.а

# **Глава 2. Структура транслятора**

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

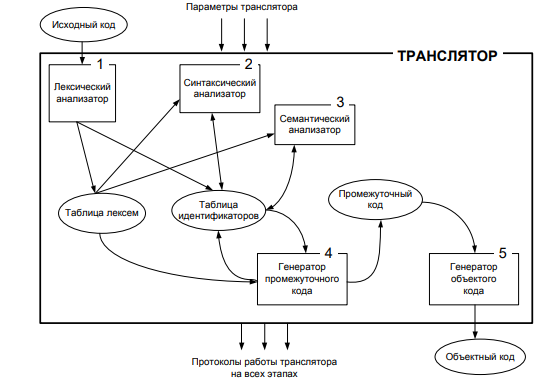


Рисунок 2.1 Структура транслятора

Первым этапом трансляции является лексический анализ, поэтому вначале исходный код подаётся на вход лексического анализатора. Лексический анализатор проверяет входные символы на допустимость и разбивает текст на токены. После чего формирует таблицу лексем и таблицу идентификаторов, которые, в свою очередь, подаются на вход синтаксического анализатора. Если исходный код написан синтаксически верно (сохранена правильность всех конструкций языка), начинается следующая фаза трансляции – семантический анализ. В противном случае работа транслятора останавливается. Семантический анализатор представлен набором функций, проверяющих соблюдение неформальных правил языка – объявление переменной до ее использования, соответствие фактических и формальных параметров функции и т.д. Последним этапом является генерация кода, во время исполнения которого формируется объектный код.

# **2.2 Перечень входных параметров транслятора**

Таблица 2.1 Входные параметры транслятора

|  |  |  |
| --- | --- | --- |
| Параметр | Тип | Характеристика |
| -in: | Обязательный | Указывает на файл с исходным кодом. Исходный код содержится в файле с расширением \*.txt |
| -out: | Не обязательный | Указывает на файл для вывода необходимых данных. Если не указан явно то имя файла формируется путем конкатенации имени файла исходного кода и постфикса «.out» |
| -log: | Не обязательный | Указывает на файл для вывода протокола работы программы. Если не указан явно то имя файла формируется путем конкатенации имени файла исходного кода и постфикса «.log» |

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

Таблица 2.2 Протоколы, формируемые транслятором языка GAA-2020

|  |  |
| --- | --- |
| Протокол | Характеристика |
| Файл для вывода протокола работы программы (-log:) | Содержит таблицу лексем, таблицу идентификаторов, результат работы синтаксического анализатора, а также возможные сообщения об ошибках. |
| Выходной файл с расширением \*.asm | Содержит код на языке ассемблера, сгенерированный на основе исходного кода. |

# **Глава 3. Разработка лексического анализатора**

# **3.1 Структура лексического анализатора**

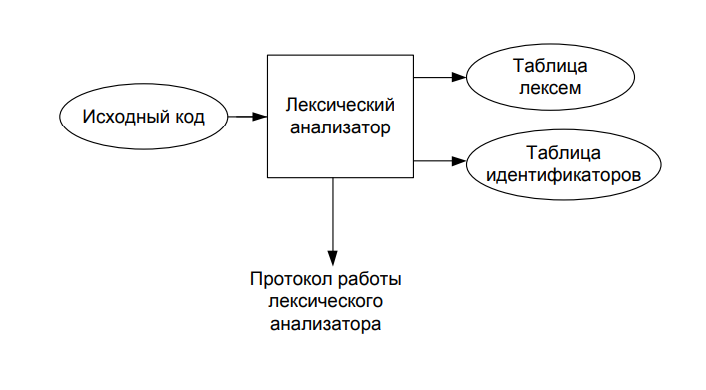
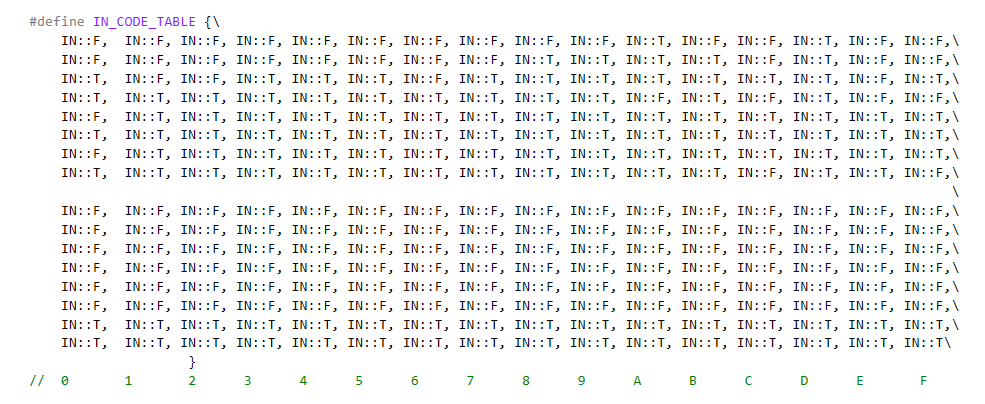


Рисунок 3.1 Структура лексического анализатора

Входными данными является исходный код на языке GAA-2020. Таблица лексем и таблица идентификаторов, сформированные в результате лексического анализа, являются результатом работы лексического анализатора.

# **3.2 Контроль входных символов**

****Рисунок 3.2 Таблица входных символов

Принцип работы таблицы заключается в соответствии значения каждого её элемента значению в таблице ASCII.

Значения символов: F – запрещенные символы, Т – разрешенные символы.

При считывании из файла символы сравниваются с символами данной таблицы. Если в таблице входной символ помечен как Т, то программа считывает его и записывает в буфер, если помечен как F – работа транслятора останавливается и выдаётся соответствующее сообщение об ошибке.

# **3.3 Удаление избыточных символов**

Избыточными символами являются пробелы, символы табуляции, символы перехода на новую строку.

Алгоритм удаления избыточных символов:

1. Посимвольно считываем файл с исходным кодом;
2. При нахождении избыточного символа пропускаем его, не записывая в результирующий массив;
3. Продолжаем посимвольное считывание файла до встречи с символом, отличным от избыточного.

# **3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем**

Таблица 3.1 Перечень ключевых слов, сепараторов, символов операций

|  |  |
| --- | --- |
| Токен | Лексема |
| integer | t |
| string | t |
| function | f |
| print | p |
| return | r |
| until | u |
| head | h |
| new | n |
| +-\*/% | v |
| = | = |
| ( | ( |
| ) | ) |
| { | { |
| } | } |
| , | , |
| ~ | ~ |
| идентификатор | i |
| литерал | l |

Фрагмент графов переходов конечных автоматов, соответствующие регулярным выражениям приведены в приложении Б.

# **3.5 Основные структуры данных**

Основные структуры данных лексического анализа приведены в приложении В.

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

Таблица 3.2 Перечень сообщений лексического анализатора

|  |  |
| --- | --- |
| Код | Сообщение |
| 120 | Lex: превышен размер таблицы лексем. |
| 121 | Lex: таблица лексем переполнена. |
| 122 | Lex: превышен размер таблицы идентификаторов. |
| 123 | Lex: таблица идентификаторов переполнена. |
| 5 | Lex: лексема не распознана. |

# **3.7 Принцип обработки ошибок**

В случае обнаружения ошибки, в log-файл выводится соответствующее сообщение и транслятор прекращает свою работу. Подсчет количества ошибок не ведется.

# **3.8 Параметры лексического анализатора**

Текст кода на языке GAA-2020 подается на вход. Параметры не определяют режим работы лексического анализатора.

# **3.9 Алгоритм лексического анализа**

1. Разделение входного текста на отдельные лексемы, запись лексем в двумерный массив.
2. Распознавание каждой строки в двумерном массиве с помощью автоматов.
3. При удачном прохождении информация заносится в таблицу лексем и таблицу идентификаторов. Возврат к шагу 2.
4. При невозможности обработать лексему выводится сообщение об ошибке, работа транслятора останавливается.
5. Переход к следующему этапу трансляции.

# **3.10 Контрольный пример**

Результат работы лексического анализатора, полученный при выполнении контрольного примера представлен в приложении Г.

# **Глава 4. Разработка синтаксического анализатора**

# **4.1 Структура синтаксического анализатора**



Рисунок 4.1 Структура синтаксического анализатора

Таблицы лексем и идентификаторов являются входными данными.

Дерево разбора является выходными данными.

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

В синтаксическом анализаторе транслятора языка GAA-2020 используется контекстно-свободная грамматика , где

T – множество терминальных символов,

N – множество нетерминальных символов,

P – множество правил языка,

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.

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

N – нетерминальные символы, представленные заглавными буквами латинского алфавита.

Таблица 4.1, описывающая правила грамматики языка GAA-2020 представлена в приложении Д.

# **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 В.

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 S) |
|  | Начальное состояние магазина автомата | Символ маркера дна стека ($) |
|  | 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 синтаксического анализатора включают в себя структуру магазинного автомата и структуру грамматики Грейбах, описывающей правила языка GAA-2020. Данные структуры представлены в приложении E.

# **4.5 Описание алгоритма синтаксического разбора**

1. В магазин записывается стартовый символ.
2. На основе полученной ранее таблицы лексем формируется входная лента.
3. Запускается автомат.
4. Выбирается цепочка по первому символу, 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 магазина. Иначе возвращаемся в предыдущее сохраненное состояние и выбираем другое правило нетерминала.
6. Если в правиле встретился нетерминал – пункт 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**

Таблица 4.3 – Структура и перечень сообщений синтаксического анализатора

|  |  |
| --- | --- |
| Код | Сообщение |
| 600 | Syn: неверная структура программы |
| 601 | Syn: ошибочный оператор |
| 602 | Syn: ошибка в выражении |
| 603 | Syn: ошибка в параметрах функции. |
| 604 | Syn: ошибка в параметрах вызываемой функции |
| 605 | Ошибка синтаксического анализа |

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

Входным параметром синтаксического анализатора является таблица лексем, полученная на этапе лексического анализа, а также правила контекстно-свободной грамматики в форме Грейбах.

Выходными параметрами являются трассировка прохода таблицы лексем и правила разбора, которые записываются в файл протокола данного этапа обработки.

# **4.8 Принцип обработки ошибок**

* синтаксический анализатор перебирает все правила и цепочки правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем;
* если невозможно подобрать подходящую цепочку, то генерируется соответствующая ошибка, которая записывается в протокол работы и работа транслятора останавливается.

# **4.9 Контрольный пример**

Протокол синтаксического разбора конструкций языка из контрольного примера и полученное дерево разбора приведены а приложении Ж.

# **Глава 5. Разработка семантического анализатора**

# **5.1 Структура семантического анализатора**

Семантический анализатор состоит из набора функций для проверки правильности исходной программы. Часть ошибок семантического анализа обрабатываются на этапе лексического анализа. Но ошибки, требующие более сложной обработки (например, несоответствие типов операндов) вынесены в отдельный этап, следующий после синтаксического анализа.

# **5.2 Функции семантического анализатора**

Таблица 5.1 Функции, выполняющие проверку на этапе семантического анализа

|  |  |
| --- | --- |
| Функция | Характеристика |
| CheckParm | Проверка параметров функции на совпадение по типу и количеству |
| CheckReturnType | Проверка совпадения типа функции и типа возвращаемого значения |
| CheckAssignValue | Проверка соответствия типа переменной и присваиваемого значения |
| CheckAllowedValue | Проверка присваиваемых числовых значений на допустимость |
| CheckFunc | Проверка стандартных функций, на отсутствие объявления |

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

Таблица 5.2 Перечень сообщений семантического анализатора

|  |  |
| --- | --- |
| Код | Сообщение |
| 2 | Sem: отсутствует точка входа head |
| 3 | Sem: допустима только 1 точка входа head |
| 4 | Sem: переменная или функция должны иметь определенный тип данных |
| 6 | Sem: повторное объявление переменной запрещено |
| 130 | Sem: количество параметров функции не совпадает. |
| 131 | Sem: типы формальных и фактических параметров функции не совпадают. |
| 132 | Sem: тип функции и тип возвращаемого значения не совпадают. |
| 133 | Sem: недопустимый тип присваиваемого значения |
| 134 | Sem: операции над аргументами разных типов недопустимы. |
| 135 | Sem: число выходит за границы допустимого диапазона. |
| 136 | Sem: недопустимые операции со строками. |
| 137 | Sem: идентификатор не может быть использован до его объявления |
| 138 | Sem: стандартные функции не требуют объявления. |

# **5.4 Принцип обработки ошибок**

Все семантические ошибки являются критическими, из-за чего транслятор прекращает свою работу и в протокол работы транслятора выводится соответствующее сообщение об ошибке.

# **5.5 Контрольный пример**

Результат работы семантических функций описан в главе 8.

# **Глава 6. Преобразование выражений**

# **6.1. Выражения, допускаемые языком**

В языке GAA-2020 допустимы выражения с использованием как целочисленных, так и строковых литералов. К строковым литералам применима только операция сложения. В выражениях языка также допустимо использование вызова функции в качестве операнда. К допустимым операторам выражений относятся арифметические операторы, описанные в пункте 1.5

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

Польская запись – это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок, а так же намного более простой обработки выражений впоследствии.

Обратная польская запись – это форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. 

Алгоритм построения:

* 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.

Таблица 6.1 – Пример преобразования выражения в обратную польскую запись

|  |  |  |
| --- | --- | --- |
| Выражение | Стек | Результат |
| (x + y)\*7 |  |  |
| x + y)\*7 | ( |  |
| + y)\*7 | ( | x |
| y)\*7 | ( + | x |
| )\*7 | ( + | x y |
| \*7 |  | x y + |
| 7 | \* | x y + |
|  | \* | x y + 7 |
|  |  | x y + 7 \* |

# **6.3 Программная реализация обработки выражений**

Программная реализация обработки выражений приведена в приложении З.

# **6.4 Контрольный пример**

Таблица 6.2. Приведение выражений к ПОЛИЗ

|  |  |
| --- | --- |
| Выражение | Обратная польская запись выражения |
| i=i(l,l) | i=ill@2# |
| i=i\*(l+i)%l | i=li+i\*l% |

# **Глава 7. Генерация кода**

# **7.1 Структура генератора кода**

Генерация объектного кода — это перевод компилятором внутреннего представления исходной программы в цепочку символов выходного языка. На вход генератора подаются таблицы лексем и идентификаторов, на основе которых генерируется файл с ассемблерным кодом.



Рисунок 7.1 Структура генератора кода

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

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

Таблица 7.1 – Соответствия типов идентификаторов языка MIM-2018 и языка Ассемблера

|  |  |
| --- | --- |
| Тип идентификатора на языке GAA-2020 | Тип идентификатора на языке ассемблера |
| integer | SDWORD |
| string | Byte |
| bool | BYTE |

# **7.3 Статическая библиотека**

В языке GAA-2020 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++. Объявление функций статической библиотеки генерируется автоматически.

Вызовы стандартных функций доступны там же, где и вызов пользовательских функций. Также в стандартной библиотеке реализованы функции для манипулирования выводом, недоступные пользователю. Эти функции представлены в таблице 7.2.

Таблица 7.2 - Дополнительные функции стандартной библиотеки

|  |  |
| --- | --- |
| Функция на языке С++ | Описание |
| void outint(int i) | Функции для вывода в стандартный поток значения целочисленного идентификатора/литерала. |
| void outstr(char\* s) | Функции для вывода в стандартный поток значения строкового идентификатора/литерала. |
| char\* copystr(char\* str1, char\* str2) | Функция, копирующая значение строки str2 в строку str1. |
| char\* joinst(char\* str1, char\* str2) | Строковая функция. Возвращает результат конкатенации строк str1, str2, записанный в строку str1. |

# **7.4 Входные параметры генератора кода**

На вход генератора кода поступают таблицы лексем и идентификаторов исходного кода программы на языке GAA-2020 . Результаты работы генератора кода выводятся в файл с расширением \*.asm.

# **7.5 Контрольный пример**

Результат генерации ассемблерного кода на основе контрольного примера из приложения А приведен в приложении И.

# **Глава 8. Тестирование транслятора**

# **8.1 Тестирование лексического анализатора**

Таблица 8.1 – Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| head  {  #~  return 0~  } | "Lex: лексема не распознана. Строка 3, позиция 1." |

# **8.3 Тестирование синтаксического анализатора**

Таблица 8.2 – Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| head  {  print 1~  }~ | “Syn: неверная структура программы. Строка -1, позиция -1.” |
| integer function F(integer )  {  new integer k=5~  return k~  }~ | "Syn: ошибка в параметрах функции. Строка 1, позиция -1." |
| head  {  new integer m=1\*/8~  return 1~  }~ | "Syn: ошибка в выражении. Строка 3, позиция 7." |

# **8.3 Тестирование семантического анализатора**

Таблица 8.3 – Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| string function copystr(str1, str2)~ | "Sem: стандартные функции не требуют объявления. Строка 1, позиция 1." |
| new integer a~  new integer a~ | "Sem: повторное объявление переменной запрещено.  Строка 2, позиция 3. " |
| string fi(string n)  {  n=’p’~  return 7~  }~ | "Sem: тип функции и тип возвращаемого значения не совпадают. Строка 4 позиция -1." |

# **Заключение**

В ходе выполнения курсовой работы был разработан транслятор для языка программирования GAA-2020. Таким образом, были выполнены основные задачи данной курсовой работы:

* сформулирована спецификация языка GAA-2020;
* разработаны конечные автоматы и алгоритмы для реализация лексического анализатора;
* разработана контекстно-свободная, приведённая к ослабленной нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
* разработан семантический анализатор, осуществляющий проверку смысла используемых инструкций;
* разработан транслятор с языка программирования GAA-2020 на язык Assembler;
* проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка GAA-2020 включает:

* 2 типа данных;
* поддержку операции вывода;
* возможность вызова функций стандартной библиотеки;
* наличие 5 арифметических операторов для вычисления выражений;
* оператор цикла;
* структурированная систем для обработки ошибок пользователя.

# **Приложение А**

#строка комментария;

integer function Func(integer x, integer y)

{

new integer z~

z= x\*(-7+y)%3~

return z~

}~

head

{

new integer x~

new integer y~

new integer g=1~

new string m~

new string n~

new string q~

m='hello '~

n='world '~

until(x=45~)

{

x=x+Ox17~

}~

print x~

print Func(5,122)\*4~

print 'hi'~

q=joinst(m,n)~

print q~

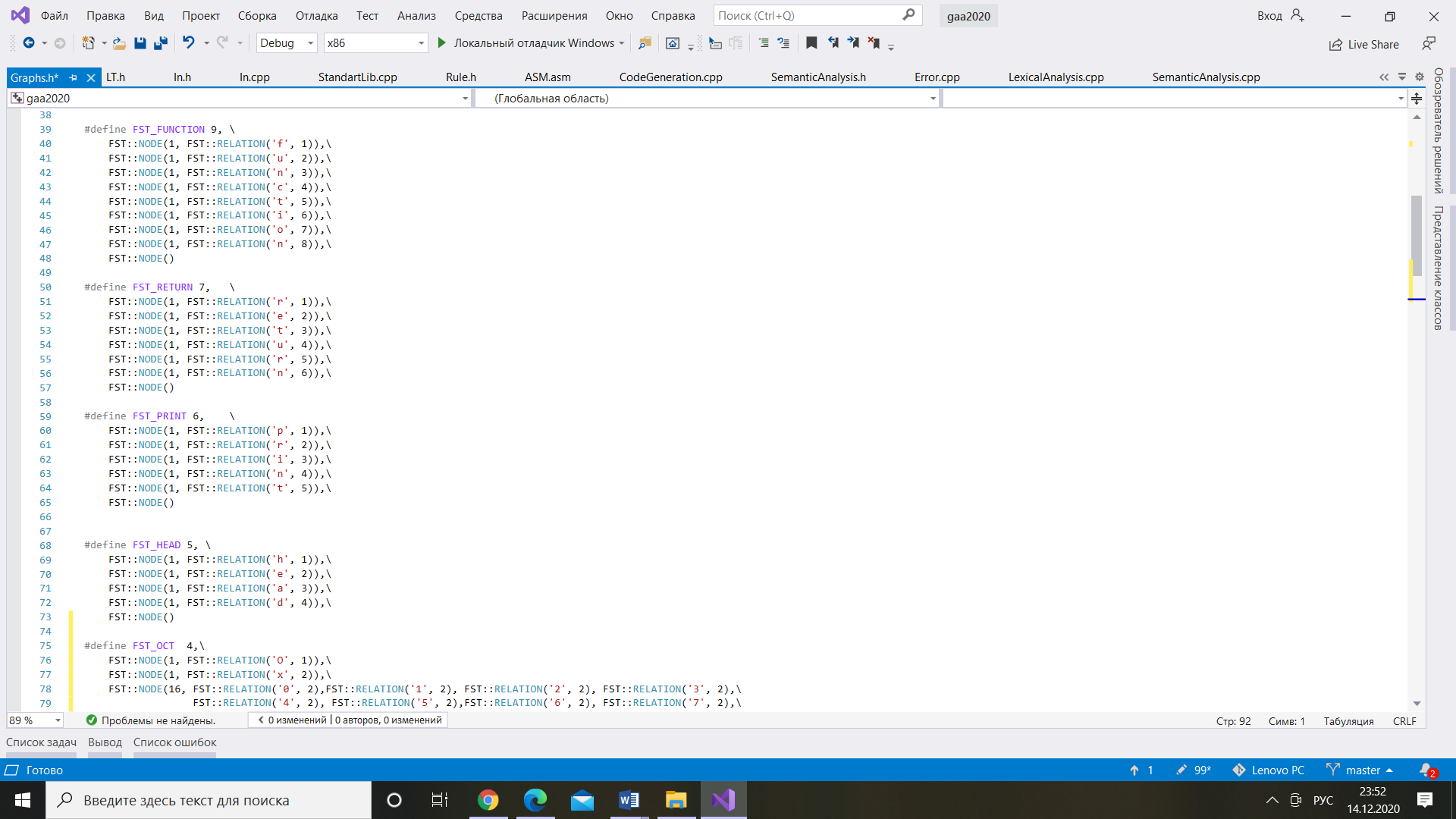
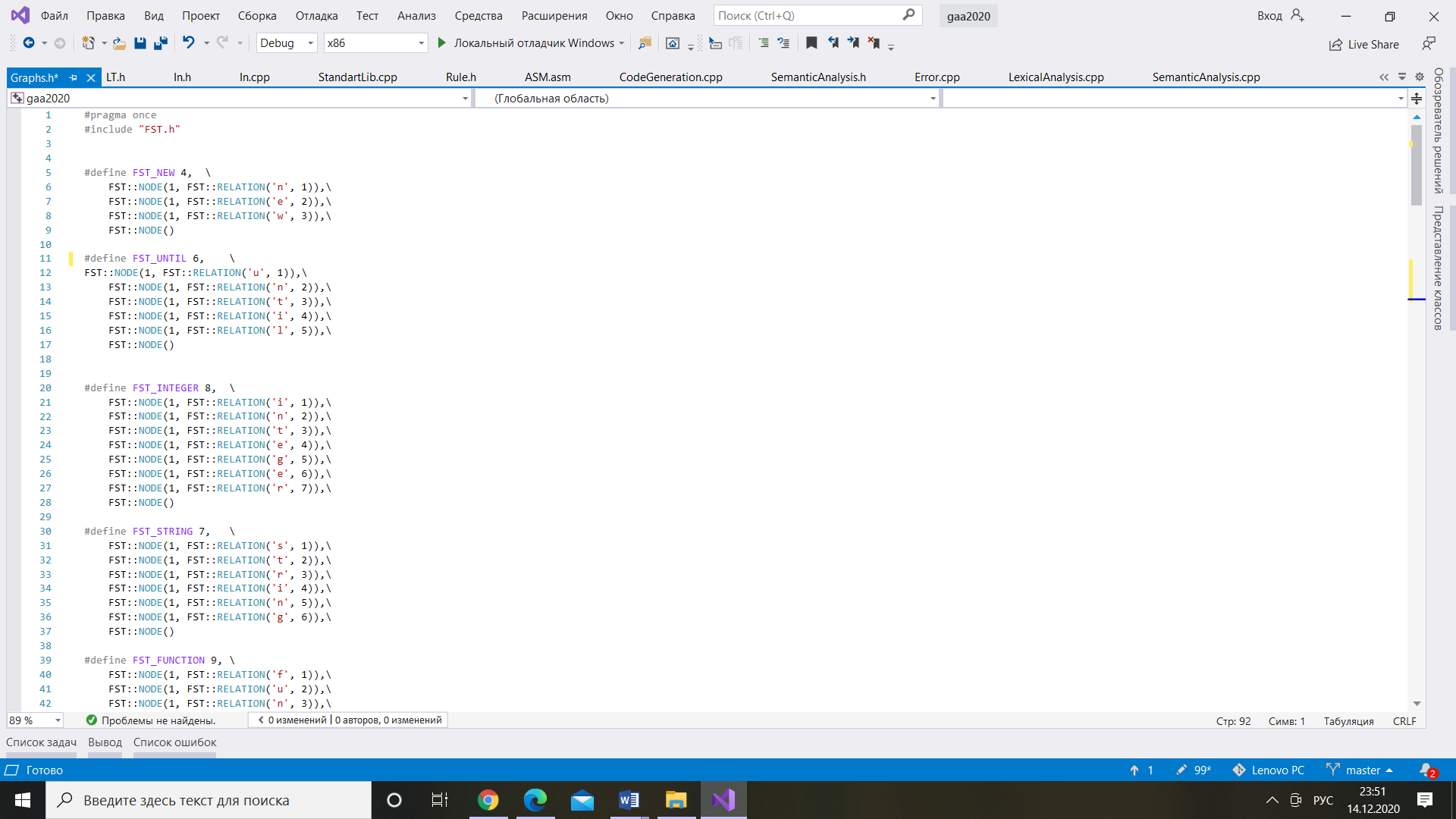
q=n+m~

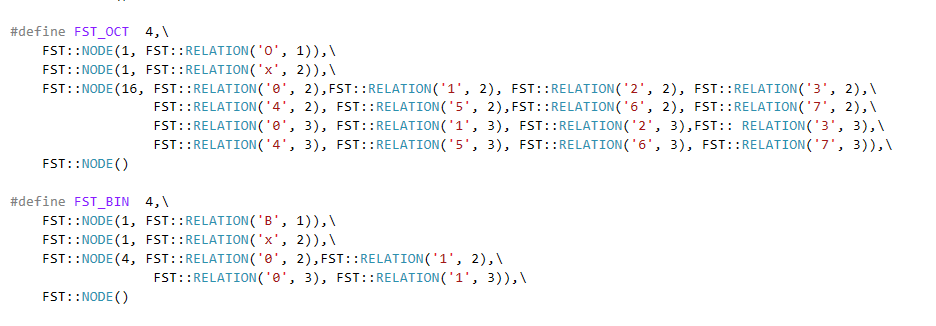
print q~

return 0~

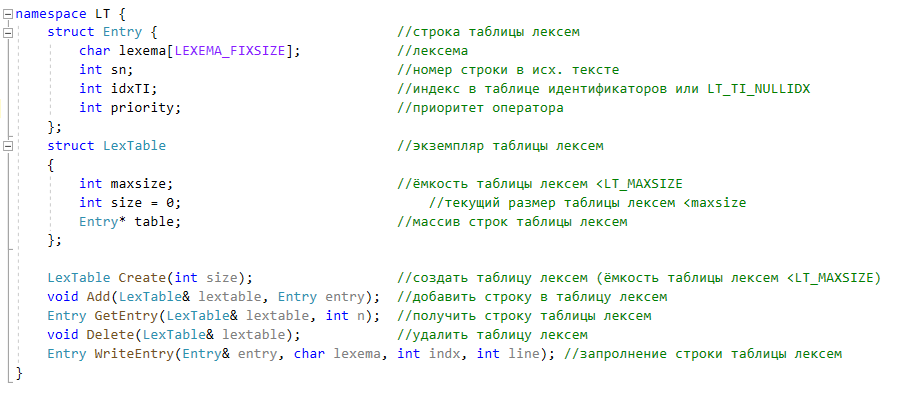
}~

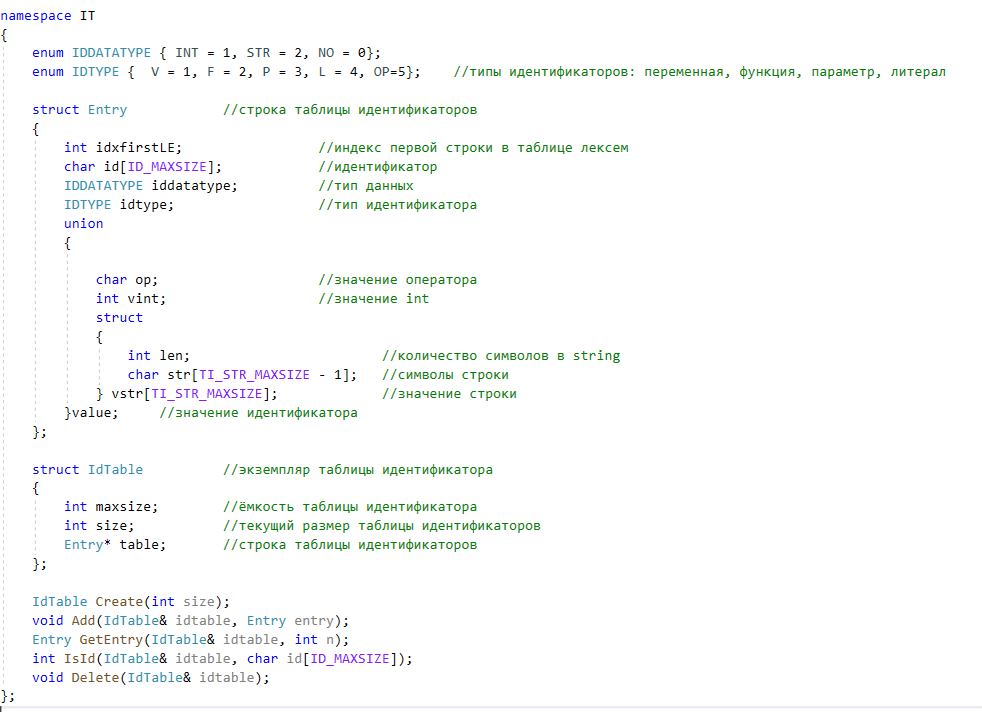
# **Приложение Б**

****



# **Приложение В**





# **Приложение Г**

Таблица лексем:

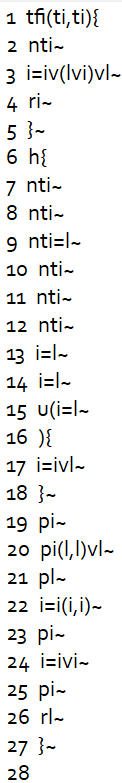


Таблица идентификаторов:

№ | Идентификатор | Тип данных | Тип идентификатора | Индекс в ТЛ | Значение

0000 Func integer функция 3 -

0001 Funcx integer параметр 6 -

0002 Funcy integer параметр 9 -

0003 Funcz integer переменная 14 0

0004 OP1 - оператор 18 \*

0005 L1 integer литерал 21 -7

0006 OP3 - оператор 21 +

0007 OP4 - оператор 24 %

0008 L2 integer литерал 26 3

0009 headx integer переменная 37 0

0010 heady integer переменная 41 0

0011 headg integer переменная 45 0

0012 L3 integer литерал 47 1

0013 headm string переменная 51 [0] ""

0014 headn string переменная 55 [0] ""

0015 headq string переменная 59 [0] ""

0016 L4 string литерал 63 [6] "hello "

0017 L5 string литерал 67 [6] "world "

0018 L6 integer литерал 73 45

0019 L7 integer литерал 80 15

0020 L8 integer литерал 90 5

0021 L9 integer литерал 92 122

0022 L10 integer литерал 94 4

0023 L11 string литерал 97 [2] "hi"

0024 joinst string функция 101 -

# **Приложение Д**

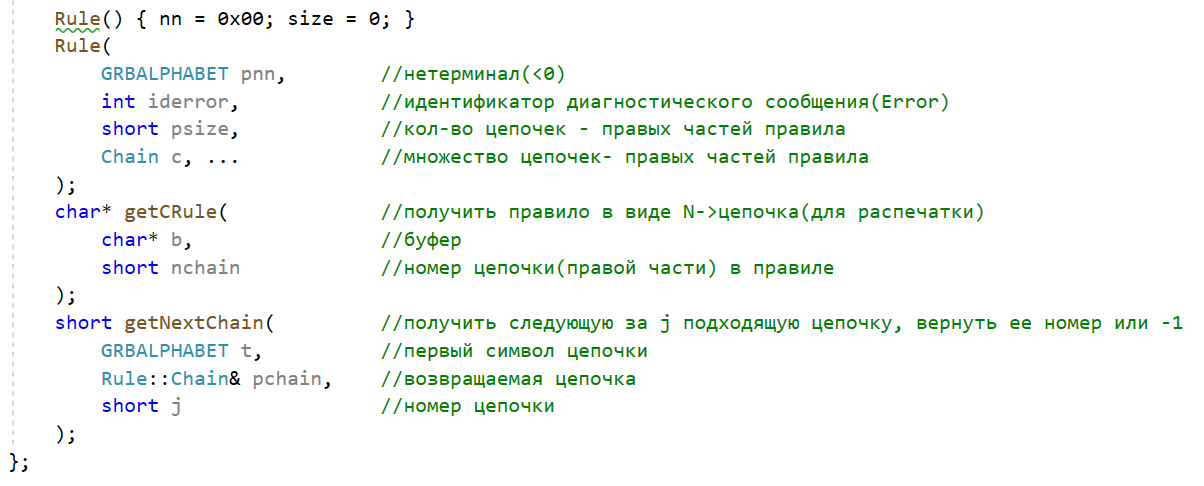
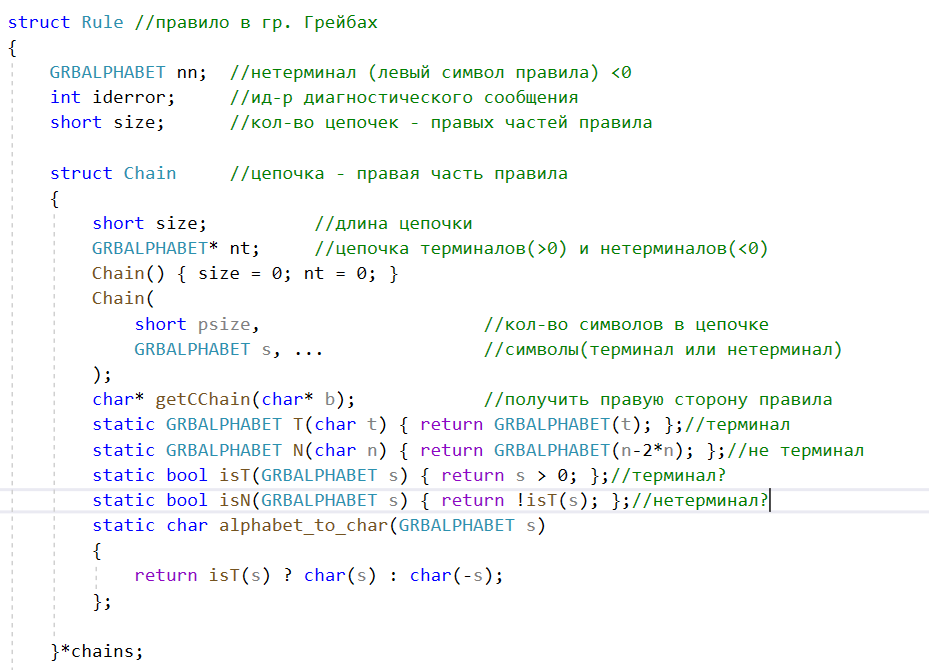
Таблица 4.1 Таблица правил переходов нетерминальных символов

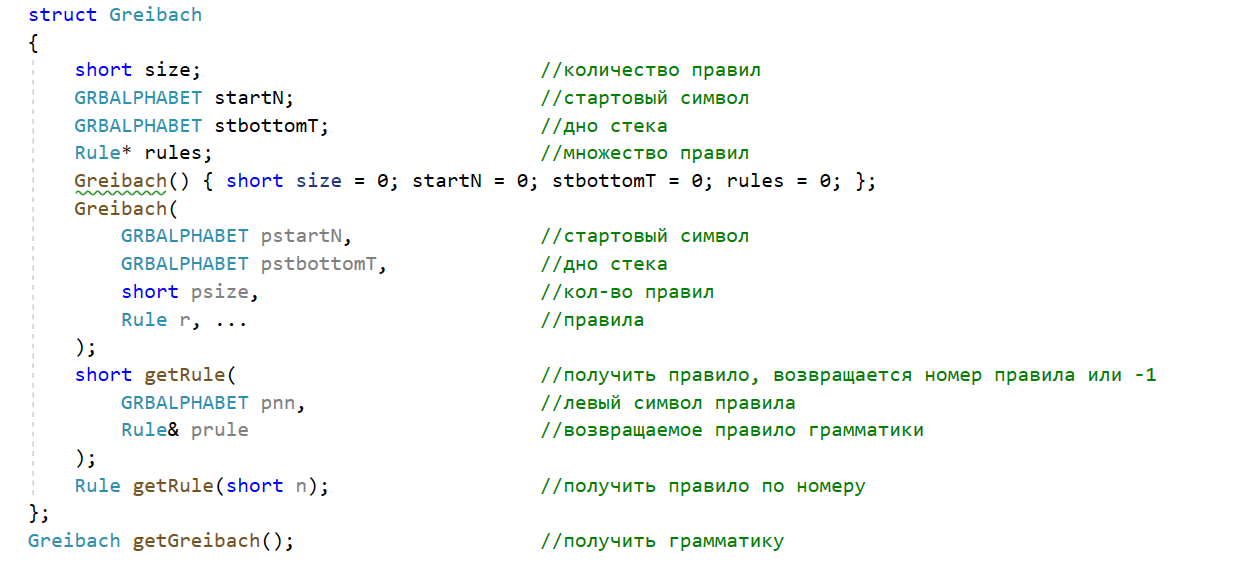
|  |  |  |
| --- | --- | --- |
| Символ | Правила | Какие правила порождает |
| S | S->h{NrE~}~  S->h{NrE~}~S  S-> tfi(F){NrE~}~  S->tfi(F){NrE~}~S | Стартовые правила, описывающее общую структуру программы |
| N | N->nti~  N->nti~N  N->nti=E~  N->nti=E~N  N->i=E~  N->i=E~N  N->ntfi(F)~  N->ntfi(F)~N  N->pi~  N->pi~N  N->pl~  N->pl~N  N->pE~  N->pE~N  N->rE~  N->rE~N  N->u(i=E~){N}~  N-> u(i=E~){N}~N | Правила для конструкций в функциях |
| E | E->i(F)  E->i(F)M  E->i(F)ME  E->i  E->l  E->(E)  E->i(W)  E->iM  E->lM  E->(E)M  E->i(W)M | Правила для выражений |
| F | F->i,F  F->i  F->l,F  F->l  F->ti  F->ti,F | Правила для параметров функций |

Продолжение таблицы 4.1 Таблица правил переходов нетерминальных символов

|  |  |  |
| --- | --- | --- |
| Символ | Правила | Какие правила порождает |
| W | W->i  W->l  W->i,W  W->l,W | Правила для параметров вызываемой функции |
| M | M->vE  M->vEM | Правила для использования операторов |

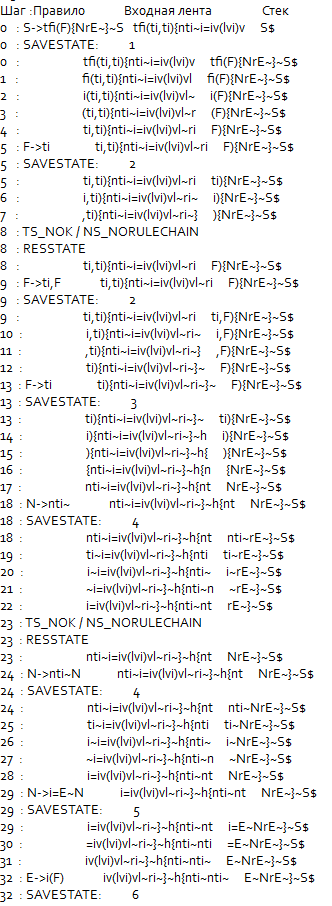
# **Приложение Е**



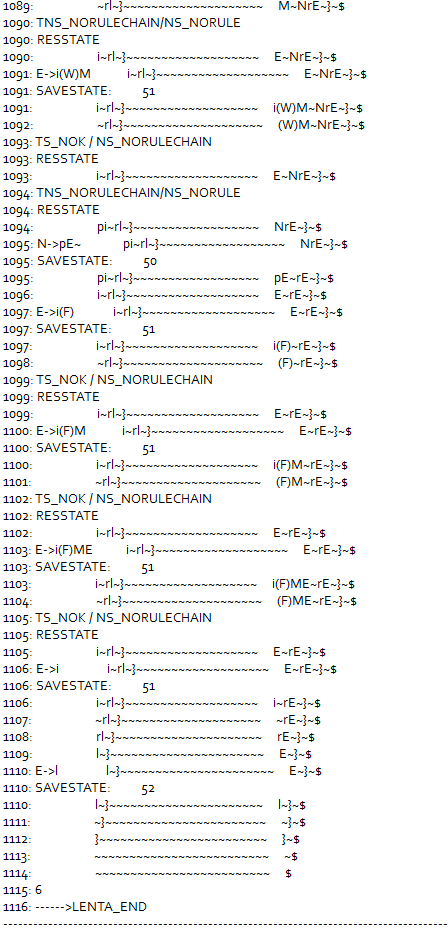


# **Приложение Ж**

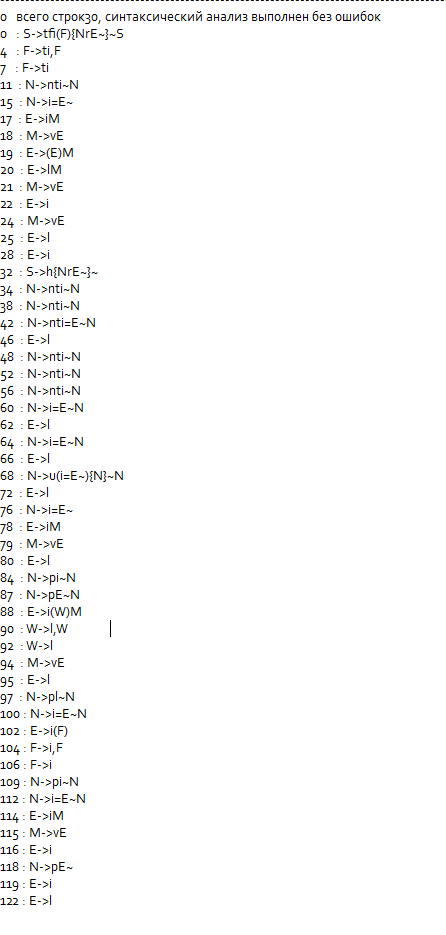
Начало разбора:



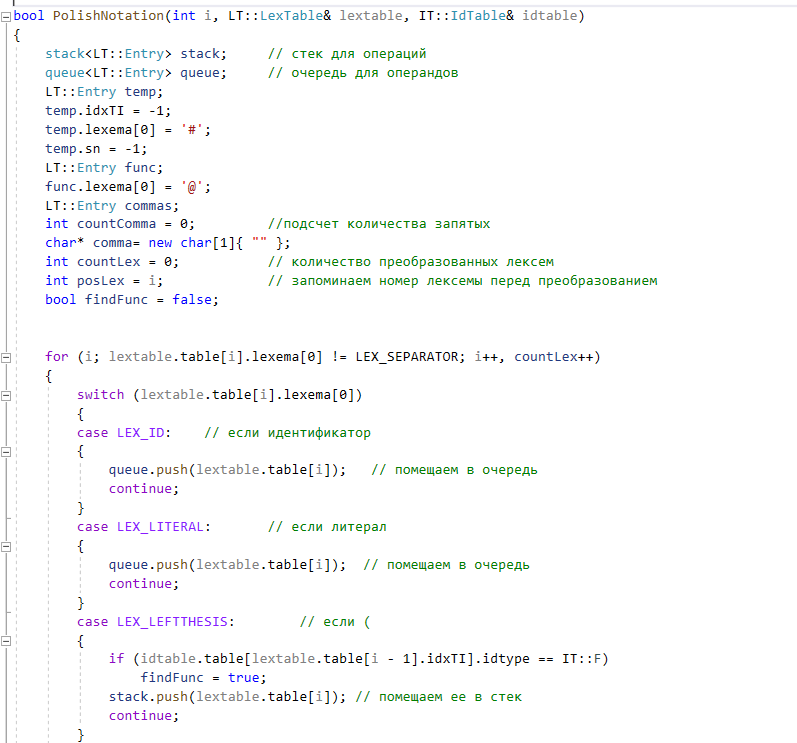
Конец разбора:

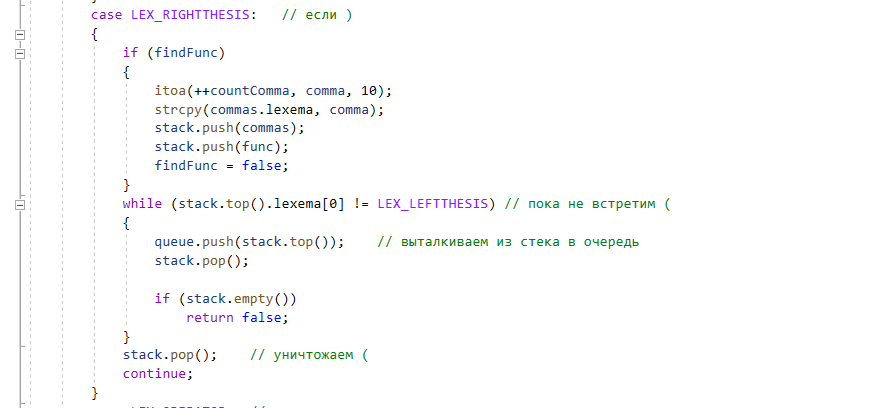


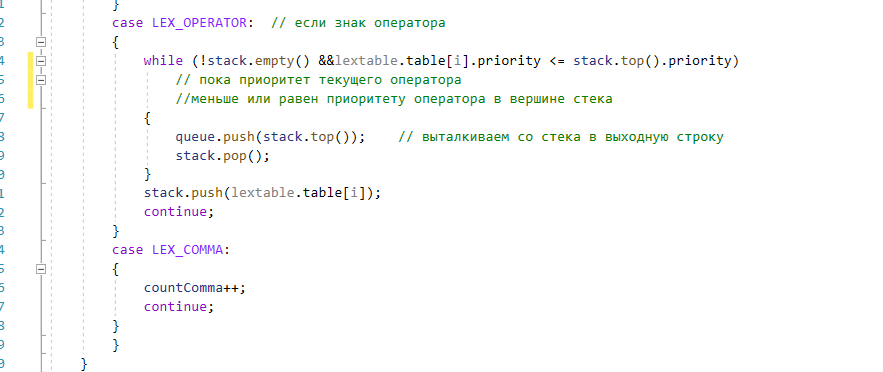
Дерево разбора:

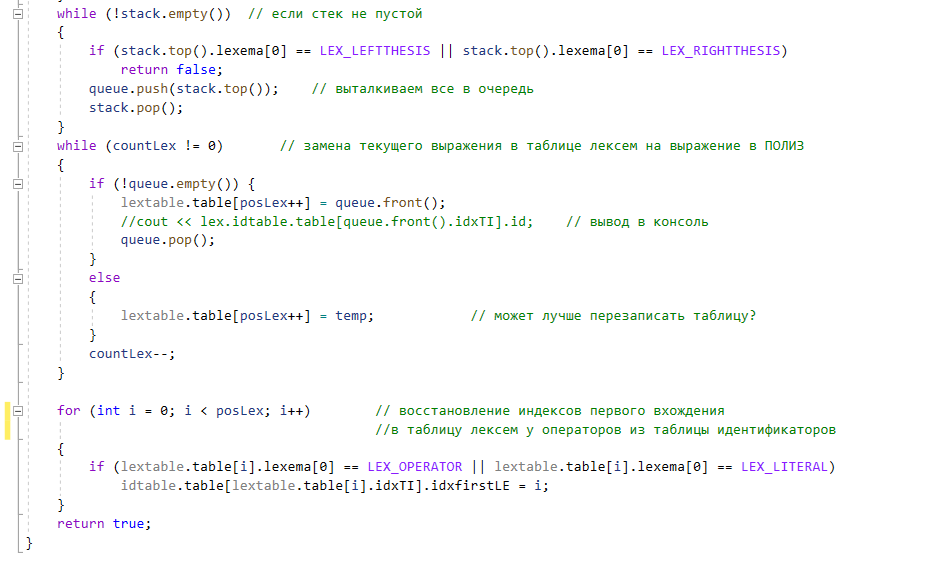


# **Приложение З**









# **Приложение И**

