**Содержание**

[**Введение** 5](#_Toc501542953)

[**1. Спецификация языка программирования** 6](#_Toc501542954)

[**1.1.** **Характеристика языка программирования** 6](#_Toc501542955)

[**1.2.** **Алфавит языка** 6](#_Toc501542956)

[**1.3.** **Символы сепараторы** 7](#_Toc501542957)

[**1.4.** **Применяемые кодировки** 7](#_Toc501542958)

[**1.5.** **Типы данных** 7](#_Toc501542959)

[**1.6.** **Преобразование типов данных** 8](#_Toc501542960)

[**1.7.** **Идентификаторы** 8](#_Toc501542961)

[**1.8.** **Литералы** 8](#_Toc501542962)

[**1.9.** **Область видимости идентификаторов** 8](#_Toc501542963)

[**1.10.** **Инициализация данных** 8](#_Toc501542964)

[**1.11.** **Инструкции языка** 8](#_Toc501542965)

[**1.12.** **Операции языка** 9](#_Toc501542966)

[**1.13.** **Выражения и их вычисления** 9](#_Toc501542967)

[**1.14.** **Программные конструкции языка** 9](#_Toc501542968)

[**1.15.** **Область видимости** 10](#_Toc501542969)

[**1.16.** **Семантические проверки** 10](#_Toc501542970)

[**1.17.** **Распределение оперативной памяти на этапе выполнения** 10](#_Toc501542971)

[**1.18.** **Стандартная библиотека и её состав** 10](#_Toc501542972)

[**1.19.** **Ввод и вывод данных** 11](#_Toc501542973)

[**1.20.** **Точка входа** 11](#_Toc501542974)

[**1.21.** **Препроцессор** 11](#_Toc501542975)

[**1.22.** **Соглашения о вызовах** 11](#_Toc501542976)

[**1.23.** **Объектный код** 11](#_Toc501542977)

[**1.24.** **Классификация сообщений транслятора** 11](#_Toc501542978)

[**1.25.** **Контрольный пример** 12](#_Toc501542979)

[**2. Структура транслятора** 13](#_Toc501542980)

[**2.1 Компоненты транслятора, их назначение и принципы взаимодействия** 13](#_Toc501542981)

[**2.2 Перечень входных параметров транслятора** 14](#_Toc501542982)

[**2.3 Перечень протоколов, формируемых транслятором и их содержимое** 14](#_Toc501542983)

[**3. Разработка лексического анализатора** 15](#_Toc501542984)

[**3.1 Структура лексического анализатора** 15](#_Toc501542985)

[**3.2 Контроль входных символов** 15](#_Toc501542986)

[**3.3 Удаление избыточных символов** 16](#_Toc501542987)

[**3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов** 16](#_Toc501542988)

[**3.5 Основные структуры данных** 17](#_Toc501542989)

[**3.6 Принцип обработки ошибок** 17](#_Toc501542990)

[**3.7 Структура и перечень сообщений лексического анализатора** 17](#_Toc501542991)

[**3.8 Параметры лексического анализатора и режимы его работы** 17](#_Toc501542992)

[**3.9 Алгоритм лексического анализа** 18](#_Toc501542993)

[**3.10 Контрольный пример** 18](#_Toc501542994)

[**4. Разработка синтаксического анализатора** 19](#_Toc501542995)

[**4.1 Структура синтаксического анализатора** 19](#_Toc501542996)

[**4.2 Контекстно свободная грамматика, описывающая синтаксис языка** 19](#_Toc501542997)

[**4.3 Построение конечного магазинного автомата** 21](#_Toc501542998)

[**4.4 Основные структуры данных** 22](#_Toc501542999)

[**4.5 Описание алгоритма синтаксического разбора** 22](#_Toc501543000)

[**4.6 Структура и перечень сообщений синтаксического анализатора** 22](#_Toc501543001)

[**4.7 Параметры синтаксического анализатора и режимы его работы** 23](#_Toc501543002)

[**4.8 Принцип обработки ошибок** 23](#_Toc501543003)

[**4.9 Контрольный пример** 23](#_Toc501543004)

[**5. Разработка семантического анализатора** 24](#_Toc501543005)

[**5.1 Структура семантического анализатора** 24](#_Toc501543006)

[**5.2 Функции семантического анализатора** 24](#_Toc501543007)

[**5.3 Структура и перечень сообщений семантического анализатора** 24](#_Toc501543008)

[**5.4 Принцип обработки ошибок** 24](#_Toc501543009)

[**5.5 Контрольный пример** 24](#_Toc501543010)

[**6. Преобразование выражений** 25](#_Toc501543011)

[**6.1 Выражения, допускаемые языком** 25](#_Toc501543012)

[**6.2 Польская запись** 25](#_Toc501543013)

[**6.3 Программная реализация обработки выражений** 26](#_Toc501543014)

[**6.4 Контрольный пример** 26](#_Toc501543015)

[**7. Генерация кода** 27](#_Toc501543016)

[**7.1 Структура генератора кода** 27](#_Toc501543017)

[**7.2 Представление типов данных в оперативной памяти** 27](#_Toc501543018)

[**7.3 Алгоритм работы генератора кода** 28](#_Toc501543019)

[**8. Тестирование транслятора** 29](#_Toc501543020)

[**8.1 Тестирование фазы проверки на допустимость символов** 29](#_Toc501543021)

[**8.2** **Тестирование лексического анализатора** 29](#_Toc501543022)

[**8.3 Тестирование синтаксического анализатора** 29](#_Toc501543023)

[**8.4 Тестирование семантического анализатора** 29](#_Toc501543024)

[**Заключение** 30](#_Toc501543025)

[**Приложения** 31](#_Toc501543026)

[**Контрольный пример** 31](#_Toc501543027)

[**Приложение А** 32](#_Toc501543028)

[**Приложение В** 41](#_Toc501543029)

[**Приложение Г** 42](#_Toc501543030)

[**Приложение Е** 48](#_Toc501543031)

# **Введение**

Основной целью данной курсовой работы является разработка транслятора для языка программирования MIM-2018. Главная задача транслятора заключается в том, чтобы сделать программу, написанную языке программирования MIM-2018, понятной компьютеру. В данном курсовом проекте трансляция будет осуществляться 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 :

1. спецификация языка программирования;
2. структура транслятора;
3. лексический анализатор;
4. синтаксический анализатор;
5. семантический анализатор;
6. преобразование выражений;
7. генерация кода;
8. тестирование транслятора.

Язык программирования MIM-2018 предназначен для работы с консолью, выполнения простейших арифметический действий, операций над строками и логическими переменными.

# **1. Спецификация языка программирования**

## **Характеристика языка программирования**

Язык MIM-2018 ­– это процедурный, универсальный, строго типизированный, компилируемый язык. He является объектно-ориентированным.

* 1. **Алфавит языка**

Исходный код MIM-2018 может содержать символы латинского алфавита, цифры десятичной системы счисления от 0 до 9, русские символы разрешены только в строковых литералах.

## **Символы сепараторы**

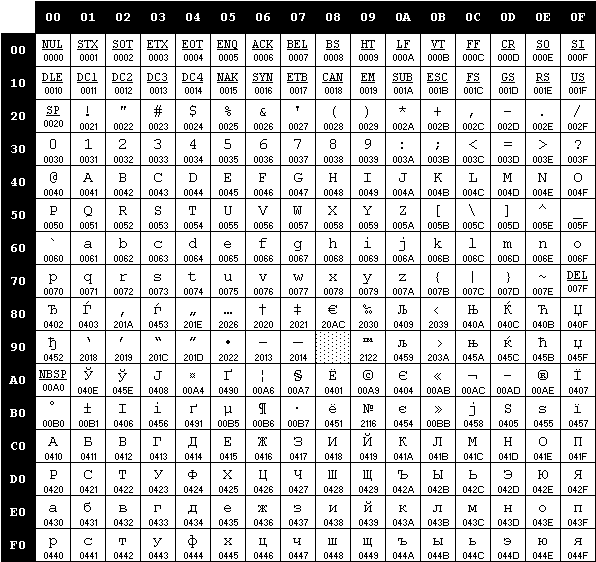
Символы, которые являются сепараторами представлены в таблице 1.1.

Таблица 1.1 – Сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение |
| ;  ‘ ‘ (пробел)  ,  >  <  =  +  -  \*  / | Разделение инструкций |
| {  } | Программный блок инструкций |
| (  ) | Параметры функций / приоритетность операций (в выражениях) |

## **Применяемые кодировки**

Для написания исходного кода на языке программирования MIM-2018 используется кодировка Windows-1251, представленная на рисунке ниже:



## **Типы данных**

В языке MIM-2018 есть 3 типа данных: целочисленный, строковый, булевый. Описание типов данных, предусмотренных в данным языке представлено в таблице 1.2.

Таблица 1.2 – Типы данных языка MIM-2018

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| int | Фундаментальный тип данных. Предусмотрен для объявления целочисленных положительных данных.  Автоматически инициализируется нулевым значением. |
| str | Фундаментальный тип данных. Предусмотрен для объявления строк. Автоматическая инициализация строкой нулевой длины. Максимальное количество символов в строке – 255; |
| bool | Фундаментальный тип данных. Предусмотрен для объявления логической переменной, которая имеет одно из двух значений: true, false. Автоматическая инициализация значением false. |

## **Преобразование типов данных**

В языке программирования MIM-2018 преобразование типов данных не поддерживается.

## **Идентификаторы**

В имени идентификатора допускаются символы латинского алфавита верхнего и нижнего регистра, но в начале имени каждого идентификатора переменной обязан стоять латинский символ «i», для идентификатора функций - «f». Максимальная длина имени - 100 символов.

## **Литералы**

В языке существует 3 типа литералов: целого, символьного, логического типов. Краткое описание литералов представлено в таблице 1.3.

Таблица 1.3 – Описание литералов

|  |  |
| --- | --- |
| Тип литерала | Описание |
| Литералы целого типа | Целочисленные неотрицательные литералы, инициализируются 0. Литералы только rvalue. |
| Строковые литералы | Символы, заключённые в ' '(одинарные кавычки), инициализируются пустой строкой, строковые переменные. Только rvalue. |
| Логические литералы | Логический литерал, true/false. Только rvalue. |

## **Область видимости идентификаторов**

Область видимости «сверху вниз» (по принципу С++). В языке MIM-2018 требуется обязательное объявление переменной перед её использованием. Имеется возможность объявления одинаковых переменных в разных блоках. Каждая переменная получает префикс – название функции, в которой она объявлена.

## **Инициализация данных**

Таблица 1.4 – Способы инициализации переменных

|  |  |
| --- | --- |
| Вид инициализации | Примечание |
| <тип данных> i<идентификатор>; | Автоматическая инициализация: переменные типа int инициализируются нулём, переменные типа str – пустой строкой, bool – true/false; |
| i<идентификатор> = <значение>; | Присваивание переменной значения. |

## **Инструкции языка**

Все возможные инструкции языка программирования MIM-2018 представлены в общем виде в таблице 1.5.

Таблица 1.5 – Инструкции языка программирования MIM-2018

|  |  |
| --- | --- |
| Инструкция | Запись на языке MIM-2018 |
| Объявление переменной | <тип данных> i<идентификатор>; |
| Присваивание | <идентификатор> = <значение> | <идентификатор>; |
| Объявление внешней функции | <тип данных> f<идентификатор>(<тип данных> <идентификатор>, …) {…} |
| Блок инструкций | {  …  } |
| Возврат из подпрограммы | get <идентификатор> | <литерал> | <выражение>; |
| Вывод данных | write (<идентификатор> | <литерал> | <выражение>); |
| Оператор цикла | go(<условие>)  {  } |

## **Операции языка**

Язык программирования MIM-2018 может выполнять арифметические операции, представленные в таблице 1.6.

Таблица 1.6 – Приоритетности операций языка программирования MIM-2018

|  |  |
| --- | --- |
| Операция | Приоритетность операции |
| (  ) | 0 |
| , | 1 |
| \*  / (деление с остатком) | 3 |
| +  - | 2 |

Максимальным значением приоритетности является “3”, минимальным “0” соответственно.

## **Выражения и их вычисления**

Круглые скобки могут использоваться для записи математических выражений, обязаны использоваться при объявлении функции, либо для составления сложных выражений в контексте условий сравнения. Фигурные скобки используются для составления блоков кода функций, операторов.

## **Программные конструкции языка**

Ключевые программные конструкции языка программирования MIM-2018 представлены в таблице 1.7.

Таблица 1.7 – Программные конструкции языка MIM-2018

|  |  |
| --- | --- |
| Главная функция (точка входа в приложение) | fmain  {  } |
| Функция | <тип> f<идентификатор>(<тип> i<идентификатор>, …)  {  …  get <выражение>;  } |

## **Область видимости**

В языке MIM-2018 переменные могут находится как внутри программного блока функций, так и вне их(при условии, что они объявлены до точки входа fmain).

Так же есть следующие ограничения:

* переменная, находящаяся в глобальной области видимости может быть объявлена, либо объявлена с инициализацией. Присвоить значение переменной, без объявления в этой же строке нельзя. Далее переменную можно использовать во всех функциях, описанных ниже ее объявления;
* вызов функций в глобальной области видимости запрещен;
* строковая переменная, объявленная в глобальной области видимости является своего рода константой и не может быть изменена.

## **Семантические проверки**

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

Таблица 1.8 – Семантические проверки

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Идентификаторы не должны повторяться |
| 2 | Тип данных переменной должен совпадать с типом значения, которое присваивается этому типу |
| 3 | Идентификатор должен быть объявлен до его использования. |
| 4 | Операнды в арифметическом или логическом выражениях не могут быть разных типов |
| 5 | Тип данных возвращаемой функции должен совпадать с типом при ее объявлении |
| 6 | Невозможно использование строк в выражениях |

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

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

## **Стандартная библиотека и её состав**

Функции стандартной библиотеки с описанием представлены в таблице 1.9. Стандартная библиотека написана на языке программирования C++.

Таблица 1.9 – Состав стандартной библиотеки

|  |  |  |  |
| --- | --- | --- | --- |
| Имя функции | Возвращаемое значение | Принимаемые параметры | Описание |
| fGetLen | int | str s - строка | Возвращает длину строки s |
| fGetMax | int | int x – число,  int y – число | Возвращает максимальное из двух значений |

## Так же, в стандартную библиотеку входит две системных функции ToConsoleNUM и ToConsoleSTR, предназначенные для вывода чисел и строк на консоль соответственно. Функции принимают по одному параметру: число, либо строку соответственно.

## **Ввод и вывод данных**

Ввод данных не поддерживается языком программирования MIM-2018.

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

## **Точка входа**

Точкой входа является функция fmain.

## **Препроцессор**

Предусмотрено подключение пользовательских библиотек через директиву: import ‘example.txt’.

## **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.

## **Объектный код**

MIM-2018 транслируется в язык ассемблера.

## **Классификация сообщений транслятора**

В случае возникновения ошибки в коде программы на языке MIM-2018 и выявления её транслятором в текущий файл протокола выводится сообщение. Их классификация сообщений приведена в таблице 1.10.

Таблица 1.10. – Классификация сообщений транслятора

|  |  |
| --- | --- |
| Интервал | Описание ошибок |
| 0 – 100 | Системные ошибки, ошибки параметров |
| 200 – 300 | Ошибки лексического анализа |
| 600 - … | Ошибки синтаксического анализа |
| 300 – 600 | Ошибки семантического анализа |

## **Контрольный пример**

Контрольный пример представлен в главе Приложения.

# **2. Структура транслятора**

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

Транслятор преобразует программу, написанную на языке MIM-2018 в программу на языке ассемблера. Компонентами транслятора являются лексический, синтаксический и семантический анализаторы, а также генератор кода на язык ассемблера. Принцип их взаимодействия представлен на рисунке 2.1.

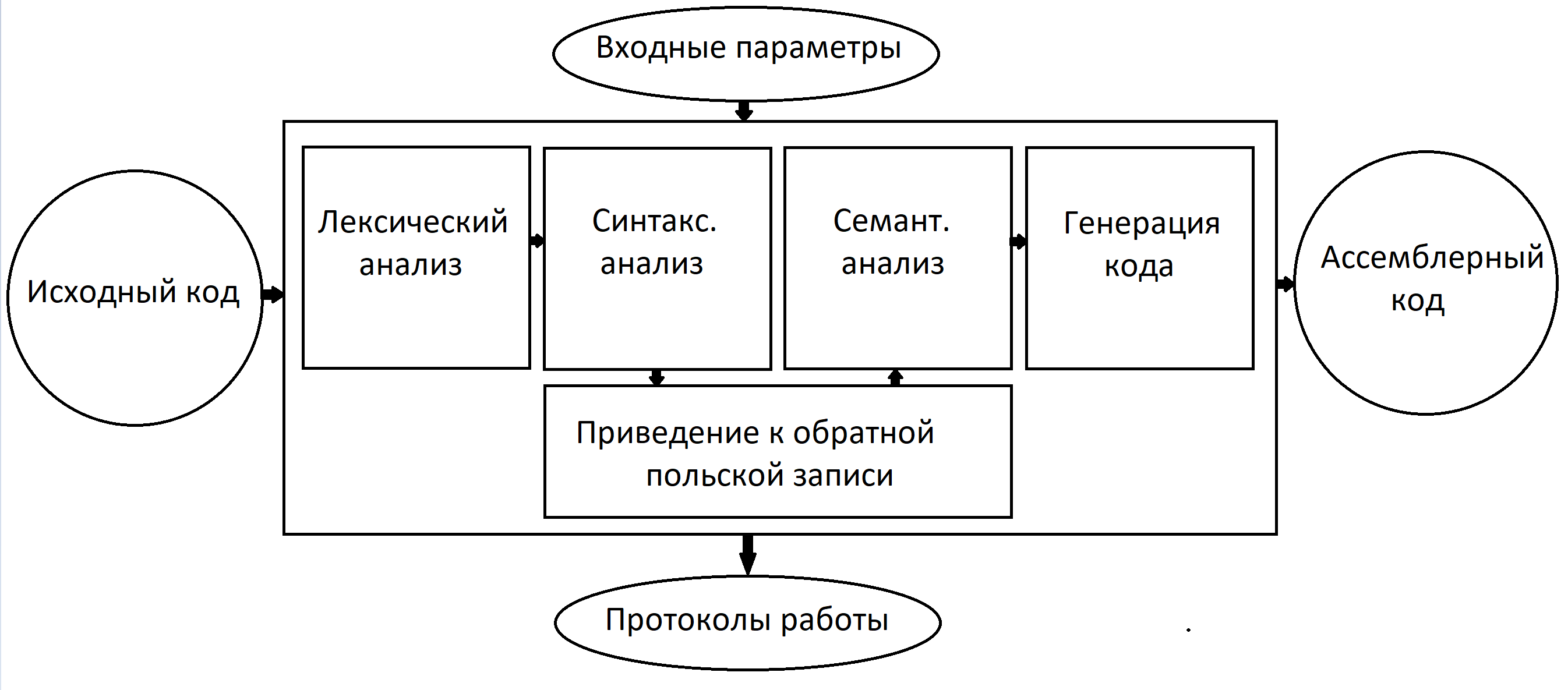


Рисунок 2.1 – Структура транслятора

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

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.

Синтаксический анализ – это основная часть транслятора, предназначенная 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.

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 MIM-2018 в код на языке Ассемблера.

## **2.2 Перечень входных параметров транслятора**

Входные параметры представлены в таблице 2.1.

Таблица 2.1 – Входные параметры транслятора языка MIM-2018

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<имя\_файла> | Входной файл с расширением .txt, в котором содержится исходный код на MIM-2018 | Не предусмотрено |
| -log:<имя\_файла> | Будет опрелелять файлы, содержащие результат работы программы:  ***.log, .sintax.log, .lex.log*** | <имя\_in\_файла>.log  <имя\_in\_файла>.sintax.log  <имя\_in\_файла>.lex.log |
| -out:<имя\_файла> | Файл для записи результата работы транслятора | <имя\_in\_файла>.asm |

Стоит заметить, что имена файлов необходимо записывать с расширением.

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

Таблица с перечнем протоколов, формируемых транслятором языка

MIM-2018 и их назначением представлена в таблице 2.2

Таблица 2.2 – Протоколы, формируемые транслятором языка MIM-2018

|  |  |
| --- | --- |
| Формируемый протокол | Описание протокола |
| Файл журнала с параметром <log> | Содержит информацию о входных параметрах в приложения, этапе проверки символов на допустимость, результат работы лексического и синтаксического анализаторов. |
| Выходной файл c параметром <out> | Содержит сгенерированный код на языке Ассемблера. |

# **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 MIM-2018. На выходе формируется таблица лексем и таблица идентификаторов. Структура лексического анализатора представлена на рисунке 3.1

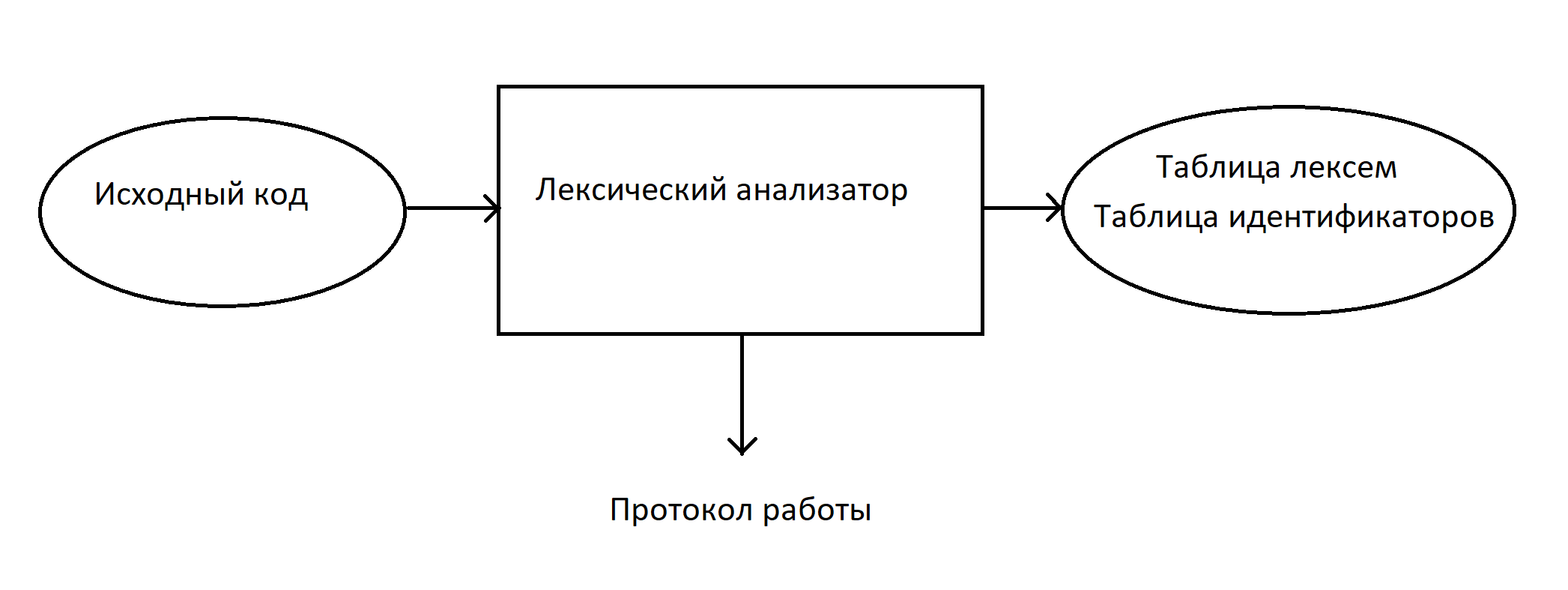


Рисунок 3.1 – Структура лексического анализатора MIM-2018

## **3.2 Контроль входных символов**

Таблица для контроля входных символов представлена на рисунке 3.2

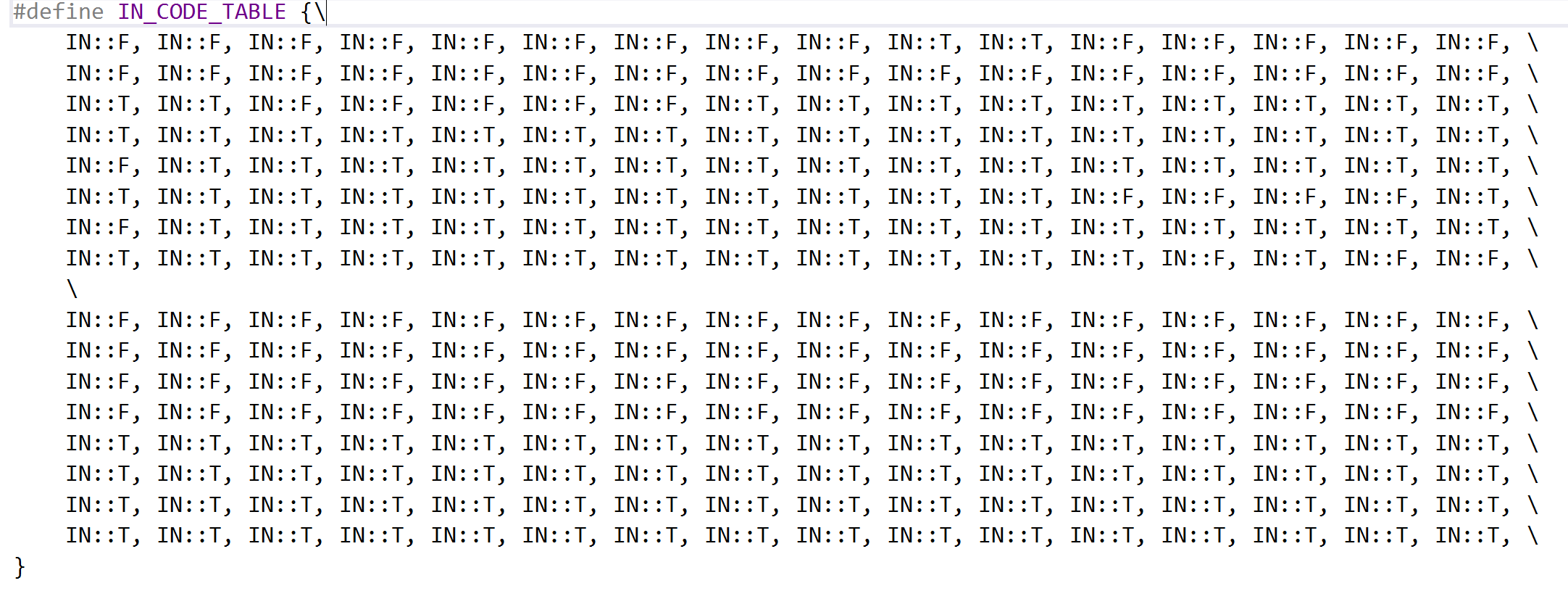


Рисунок 3.2. – Таблица контроля входных символов

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

Описание значения символов: T – разрешённый символ, F – запрещённый символ, I – игнорируемый символ.

## **3.3 Удаление избыточных символов**

Избыточными символами являются символы табуляции и пробелы.

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-разделителей, не записываем их в таблицу лексем;
4. в цикле проверяем: если следующий символ в потоке будет знаком табуляции или пробела – пропускаем этот символ до тех пор, пока следующим символом будет не \t и не ‘ ‘.

## **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. Данное соответствие описано в таблице 3.1.

Таблица 3.1 Соответствие ключевых слов, символов операций и сепараторов с лексемами

|  |  |  |
| --- | --- | --- |
| Тип цепочки | Цепочка | Лексема |
| Ключевые слова | import | p |
| int | n |
| str | s |
| bool | b |
| fmain | m |
| get | g |
| write | w |
| is | q |
| go | c |
| Иное | Идентификатор | i |
| Булевый литерал | t / f |

Продолжение таблицы 3.1

|  |  |  |
| --- | --- | --- |
| Сепараторы | ; | ; |
| , | , |
| { | { |
| } | } |
| ( | ( |
| ) | ) |
| = | = |
| Операторы | - | - |
| \* | \* |
| + | + |
| / | / |
| > | > |
| < | < |

Пример реализации таблицы лексем представлен в приложении А.

Также в приложении А находятся конечные автоматы, соответствующие лексемам языка MIM-2018.

## **3.5 Основные структуры данных**

Основные структуры таблиц лексем и идентификаторов данных языка MIM-2018, используемых для хранения, представлены в приложении А.

В таблице лексем содержатся сами лексемы, строка для каждой лексемы, в которой она была замечена. Так же размер самой таблицы лексем и строка, содержащая постфикс в виде имени функции, в которой в текущий момент производится лексический анализ.

В таблице идентификаторов содержится имя идентификатора, его номер в таблице лексем, тип данных, смысловой тип идентификатора и его значение.

## **3.6 Принцип обработки ошибок**

При возникновении ошибки работа транслятора прекращается.

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

Перечень сообщений лексического анализатора представлен ниже:

* превышен максимальный размер таблицы лексем;
* превышен максимальный размер таблицы идентификаторов;
* определения такой лексемы не существует;
* не указана крайняя кавычка строкового литерала;
* превышена максимальная длина строки;
* превышено максимальное значение размера идентификатора;
* превышено максимальное значение целого числа.

## **3.8 Параметры лексического анализатора**

Входным параметром лексического анализатора является исходный текст программы, написанный на языке MIM-2018.

## **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. Это основывается на работе конечных автоматов, которую можно представить в виде графов. Регулярные выражения — аналитический или формульный способ задания регулярных языков. Они состоят из констант и операторов, которые определяют множества строк и множество операций над ними. Любое регулярное выражение можно представить в виде графа.

Алгоритм работы:

1. считывание текста из исходного файла в буфер;
2. далее – посимвольное считывание из буфера. Если текущий символ конец строки – пункт 8;
3. проверка символа на допустимость;
4. если текущий символ сепаратор – запись в таблицу лексем. Если нет – дозапись в строку-буфер до тех пор, пока сепаратор не встретится;
5. проверка строки-буфера на одно из зарезервированных слов. Если такое есть – запись в таблицу лексем и переход к пункту 2;
6. проверка строки-буфера на соответствие одному из конечных автоматов. Если нет – пункт 8, выход из программы с ошибкой;
7. запись в таблицу лексем соответствующего значения. Запись в таблицу идентификаторов того или иного идентификатора или литерала. Переход к пункту 2;
8. конец работы анализатора.

Все регулярные выражения, имеющиеся в языке, представлены в приложении А.

## **3.10 Контрольный пример**

Результат работы лексического анализатора – таблицы лексем и идентификаторов – представлен в приложении А.

# 

# **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, полученная после фазы лексического анализа. Выходом – дерево разбора. Структура синтаксического анализатора представлена на рисунке 4.1.

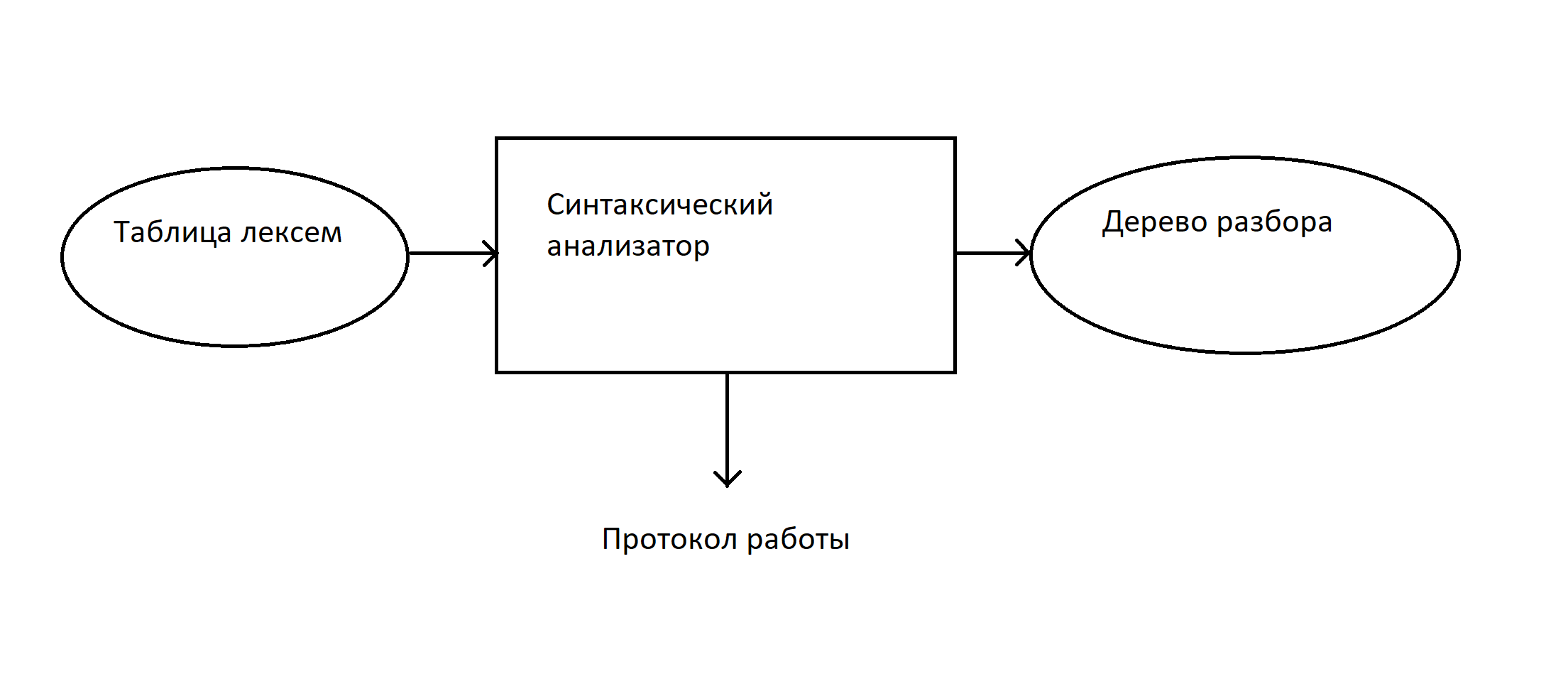


Рисунок 4.1 – Структура синтаксического анализатора

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

В синтаксическом анализаторе транслятора языка MIM-2018 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 ослабленную нормальную форму Грейбах:

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

Грамматика языка MIM-2018 представлена в приложении Б.

* TS – терминальные символы, которыми являются сепараторы, знаки арифметических операций и некоторые строчные буквы.
* NS – нетерминальные символы, представленные заглавными буквами латинского алфавита.

Таблица 4.1 – Перечень правил, составляющих грамматику языка и описание нетерминальных символов MIM-2018

|  |  |
| --- | --- |
| Нетерминал | Цепочки правил |
| S | m{N}  ni;S  si;S  bi;S  ni=E;S  si=E;S  bi=E;S  nFS  sFS  bFS  nFNS  sFNS  bFNS |
| N | ni;N  ni;  si;N  si;  bi;N  bi;  ni=E;N  ni=E;  si=E;N  si=E;  bi=i(W);N  bi=i(W);  bi=i();N  bi=i();  bi=C;N  bi=C;  i=E;N  i=E;  i=C;N  i=C;  i(W);N  i(W);  i();N  i();  q(C){N}  q(C){N}N  c(C){N}N  c(C){N}  w(E);N  w(E); |
| E | t  f  iO  i  (E)O  (E)  i(W)O  i(W)  i()O  i() |
| O | +E  -E  \*E  /E  +EO  -EO  \*EO  /EO |
| P | ni,P  ni  si,P  si  bi,P  bi |
| W | i,W  i  t,W  t  f,W  f |
| F | i(P){NG}  i(){NG}  i(P){G}  i(){G}  i(P){NG}nF  i(){NG}nF  i(P){NG}sF  i(){NG}sF  i(P){NG}bF  i(){NG}bF  i(P){G}nF  i(){G}nF  i(P){G}sF  i(){G}sF  i(P){G}bF  i(){G}bF |
| C | t  f  i < i  i > i  i = i  t = i  i = t  f = i  i = f  (E) < (E)  (E) > (E)  (E) = (E)  t = (E)  f = (E)  (E) = t  (E) = f  (E) = i  i = (E) |
| G | gE;  gE = E;  gE < E;  gE > E; |

## **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 MIM-2018. Данные структуры представлены в приложении В.

## **4.5 Описание алгоритма синтаксического разбора**

Принцип работы автомата следующий:

## В магазин записывается стартовый символ.

## На основе полученной ранее таблицы лексем формируется входная лента.

## Запускается автомат.

## Выбирается цепочка по первому символу, соответствующая нетерминальному символу, записывается в магазин в обратном порядке.

## 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.

## Если в правиле встретился нетерминал – пункт 4.

## 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 ниже:

* неверная структура программы;
* ошибочный оператор;
* ошибка выражения;
* ошибка операндов выражения;
* ошибка в параметрах определения функции;
* ошибка в параметрах вызываемой функции;
* неверная структура описания функции;
* неверное условие;
* ошибка возвращаемого значения функции.

## **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**

Обработка ошибок происходит следующим образом:

* 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 программа останавливается.

**4.9 Контрольный пример**

Пример разбора синтаксическим анализатором исходного кода на языке MIM-2018 представлен в приложении Г. Дерево разбора исходного кода также представлено в приложении Г.

# **5. Разработка семантического анализатора**

## **5.1 Структура семантического анализатора**

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 и преобразования к польской записи, так как с таблицей лексем после в форме ПОЛИЗ работать намного удобнее.

## **5.2 Функции семантического анализатора**

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

## **5.3 Перечень сообщений семантического анализатора**

Сообщения, формируемые семантическим анализатором, представлены ниже:

* превышена максимальная длина строки;
* превышено максимальное значение целого числа;
* повторное объявление идентификатора;
* не найдено объявления идентификатора;
* невозможно переопределить стандартную функцию;
* несоответствие типа возвращаемого значения функции;
* несоответствие типов в выражении;
* операции со строками в условии не возможны;
* несоответствие типов в условии;
* невозможно присвоить значение литералу;
* невозможно присвоить значение функции.

## **5.4 Принцип обработки ошибок**

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

## **5.5 Контрольный пример**

Результат работы контрольного примера расположен в приложении А, где показан результат лексического анализатора, т.к. некоторые ошибки семантического анализа обрабатываются еще на этапе формирования таблицы лексем и идентификаторов.

## **Глава 6. Преобразование выражений**

## **6.1 Выражения, допускаемые языком**

В языке MIM-2018 допускаются выражения, применимые к целочисленным типам данных, в так же выражения условий. В выражениях поддерживаются арифметические операции, такие как +, -, \*, / и (), и вызовы функций как операнды арифметических выражений. Для условных выражений те же операции, но с обязательным использованием знаков >, < или = (проверка на равенство, не присваивание), эти операции не имеют приоритета и записываются в таблицу лексем по мере их встречи в исходном файле.

Приоритет операций представлен в таблице 6.1.

Таблица 6.1 – Приоритет операций в языке MIM-2018

|  |  |
| --- | --- |
| Приоритет | Операция |
| 0 | ( |
| 0 | ) |
| 1 | , |
| 2 | + |
| 2 | - |
| 3 | \* |
| 4 | ( – скобка параметров функции |
| 4 | ) – скобка параметров функции |

## **6.2 Польская запись**

Выражения в языке MIM-2018 преобразовываются к обратной польской записи.

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:

1. посимвольно перебираем таблицу лексем;
2. если текущий символ – какой-либо оператор или идентификатор, записываем его в финальную строку лексем и переходим к пункту 1;
3. если идентификатор – функция, перед ним в финальную строку записывается токен “F”, означающий вызов функции;
4. если текущий символ – фигурная скобка, то: для открывающейся скобки в финальную строку записывается токен “B”, означающий начало процедуры, для закрывающейся – “E”, определяющий окончание процедуры;
5. открывающаяся скобка автоматически заносится в стек операций;
6. закрывающаяся скобка выталкивает все операции из стека в финальную строку и самоуничтожается вместе с открывающейся скобкой;
7. операция выталкивает все операции с большим или равным приоритетом в финальную строку;
8. когда встречается символ конца строки кода (“;”) – все операции выталкиваются из стека в финальную строку.

Таблица 6.2 – Пример преобразования выражения в обратную польскую запись

|  |  |  |
| --- | --- | --- |
| Исходная строка | Результирующая строка | Стек |
| i\*i - i(i) |  |  |
| \*i - i(i) | i |  |
| i - i(i) | i | \* |
| - i(i) | ii | \* |
| i(i) | ii\* | - |
| (i) | ii\*Fi | - |
| i) | ii\*Fi | - |
| ) | ii\*Fii | - |
|  | ii\*Fii- |  |

## **6.3 Программная реализация обработки выражений**

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

## **6.4 Контрольный пример**

Пример преобразования выражения к польской записи представлен в таблице 6.2. Преобразование выражений в формат польской записи в нашем случае необходимо для построения более простых алгоритмов при последующей обработки таблицы лексем.

В приложении Д приведены изменённые таблицы лексем и идентификаторов, отображающие результаты преобразования выражений в польский формат.

# **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.

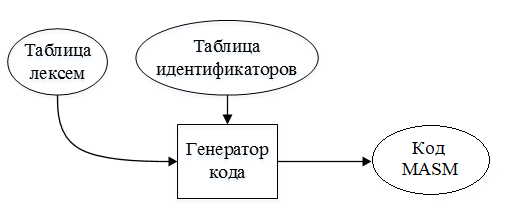


Рисунок 7.1 Структура генератора кода

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

Элементы таблицы идентификаторов расположены в разных сегментах языка ассемблера – .data и .const. Идентификаторы языка MIM-2018 размещены в сегменте данных(.data). Литералы – в сегменте констант (.const). Соответствия между типами данных идентификаторов на языке MIM-2018 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 MIM-2018 | Тип идентификатора на языке ассемблера | Пояснение |
| int | SDWORD | Хранит целочисленный тип данных со знаком. |
| str | Byte – для литералов  DWORD – для переменных | Литерал хранит последовательность байтов.  Переменные хранят указатель на начало строки литерала. |
| bool | BYTE | Хранит целочисленный тип данных, принимающий в качестве значения 0 или 1. |

## **7.3 Алгоритм работы генератора кода**

Преобразования происходят по принципу, встретив определённую лексему и зная, в каком месте программы находится сейчас лексема, программа генерирует код на языке Ассемблера.

На рисунке 7.2 представлен пример описания лексемы на языке Ассемблера.

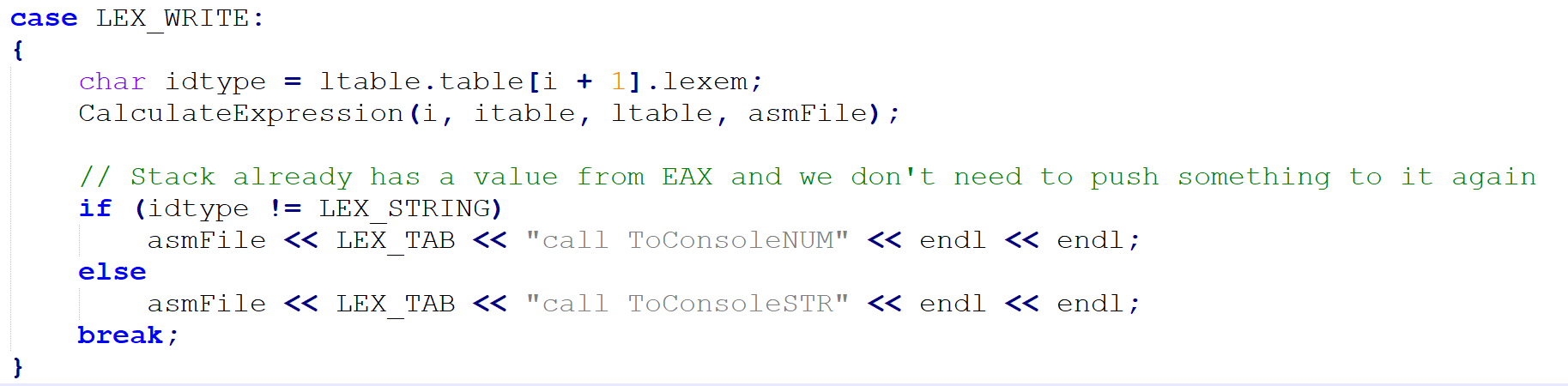


Рисунок 7.2 – Алгоритм для лексемы ‘w’

Генерируемый код записывается в файл, который был параметром ‑out. Сгенерированный код можно посмотреть в приложении Е.

# **8. Тестирование транслятора**

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

В языке MIM-2018 не разрешается использовать запрещённые входным алфавитом символы. Результат использования запрещённого символа показан в таблице 8.1.

Таблица 8.1 – Тестирование фазы проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| int @fsum(str istr){get ‘hi’;} | Ошибка 10: Недопустимый символ в исходном файле, строка: 1, символ: 5 |

## **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 |
| int variable = 2; | Ошибка 200: Определения такой лексемы не существует, строка: 1, символ: 4 |

## **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 ffunc(){} | Ошибка 606: Неверная структура описания функции, строка 1, символ: -1 |

## **8.4 Тестирование семантического анализатора**

Итоги тестирования семантического анализатора приведены в таблице 8.4.

Таблица 8.4 – Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| Int x = 10;  Int x = 1; | Ошибка 302: Повторное объявление идентификатора, строка: 2, символ: -1 |

## **Заключение**

В ходе выполнения курсовой работы был разработан транслятор для языка программирования MIM-2018. Таким образом, были выполнены основные задачи данной курсовой работы:

* Сформулирована спецификация языка MIM-2018;
* 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 MIM-2018 на язык низкого уровня Assembler;
* Проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка MIM-2018 включает:

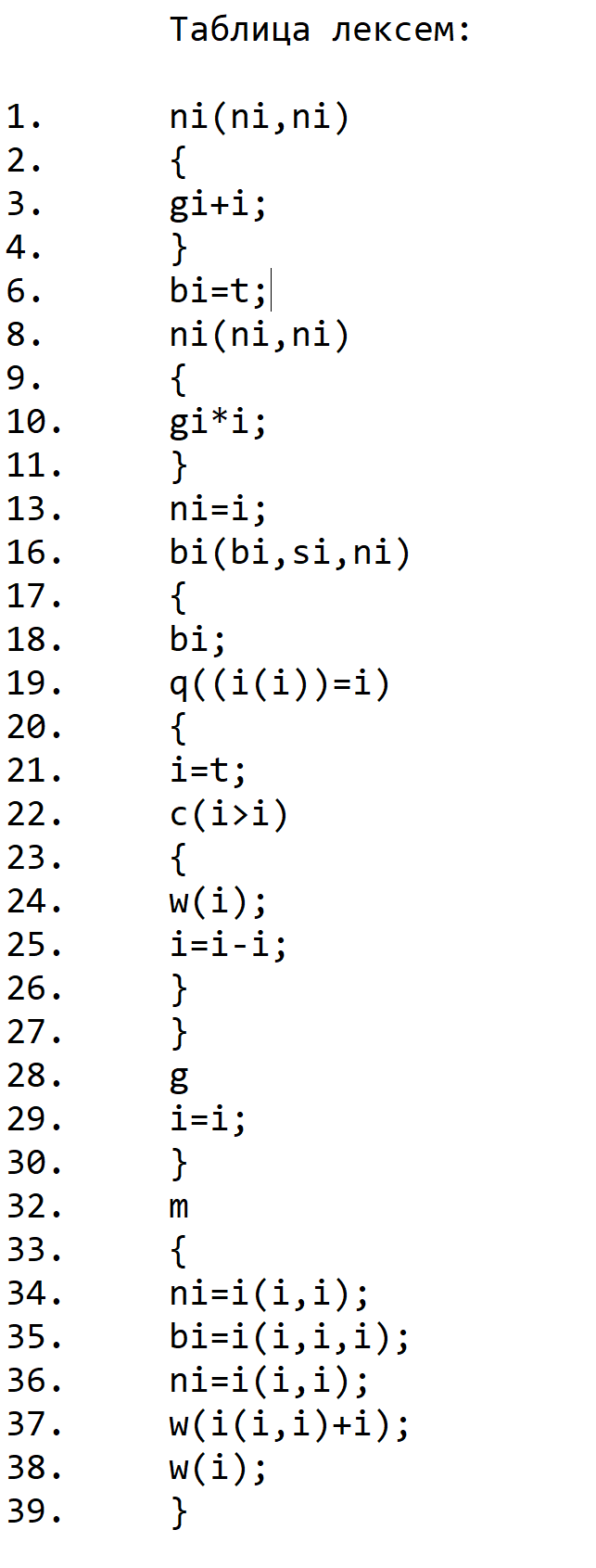
1. 3 типа данных;
2. поддержка операции вывода;
3. объявление глобальных переменных;
4. возможность вызова функций стандартной библиотеки;
5. наличие 4 арифметических операторов для вычисления выражений;
6. оператор условия и оператор цикла;
7. структурированная система для обработки ошибок пользователя.

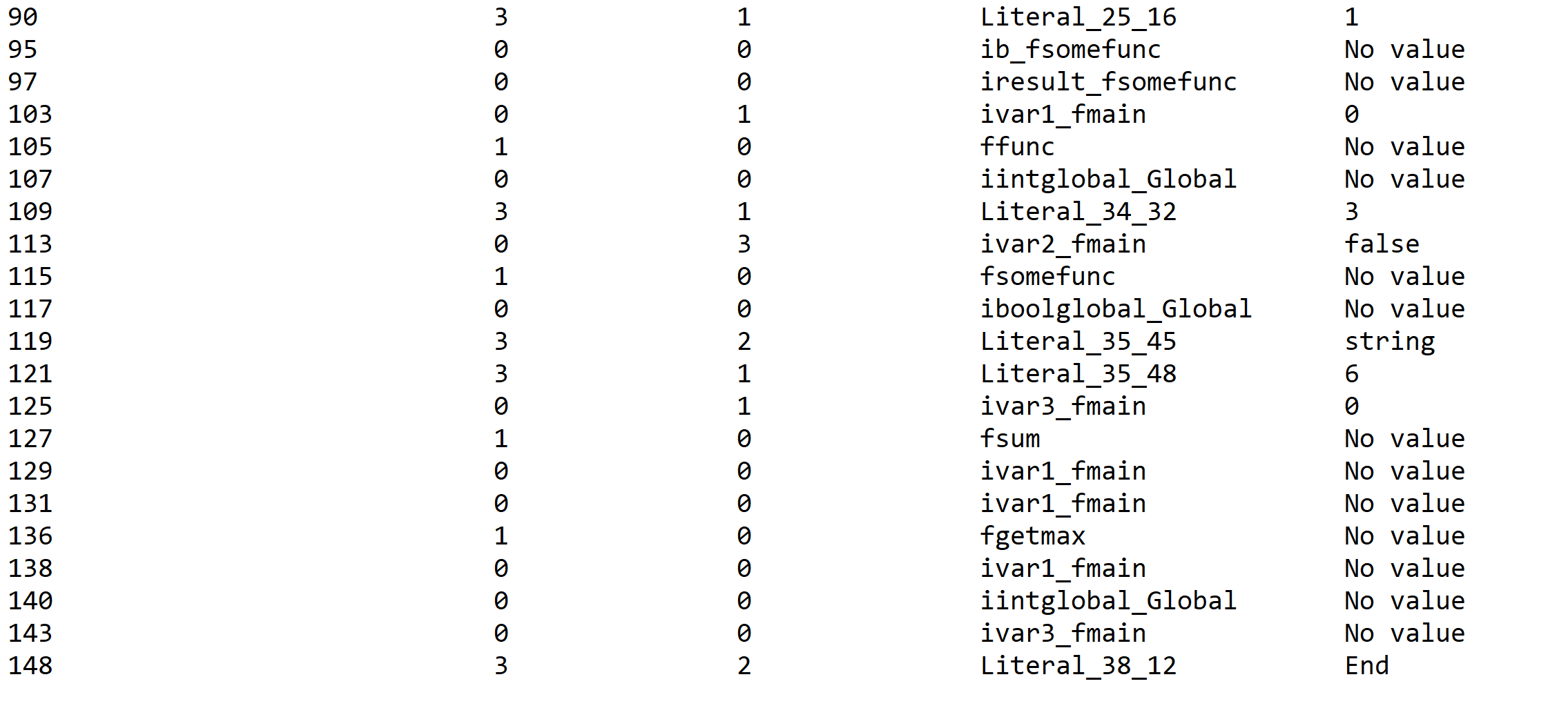
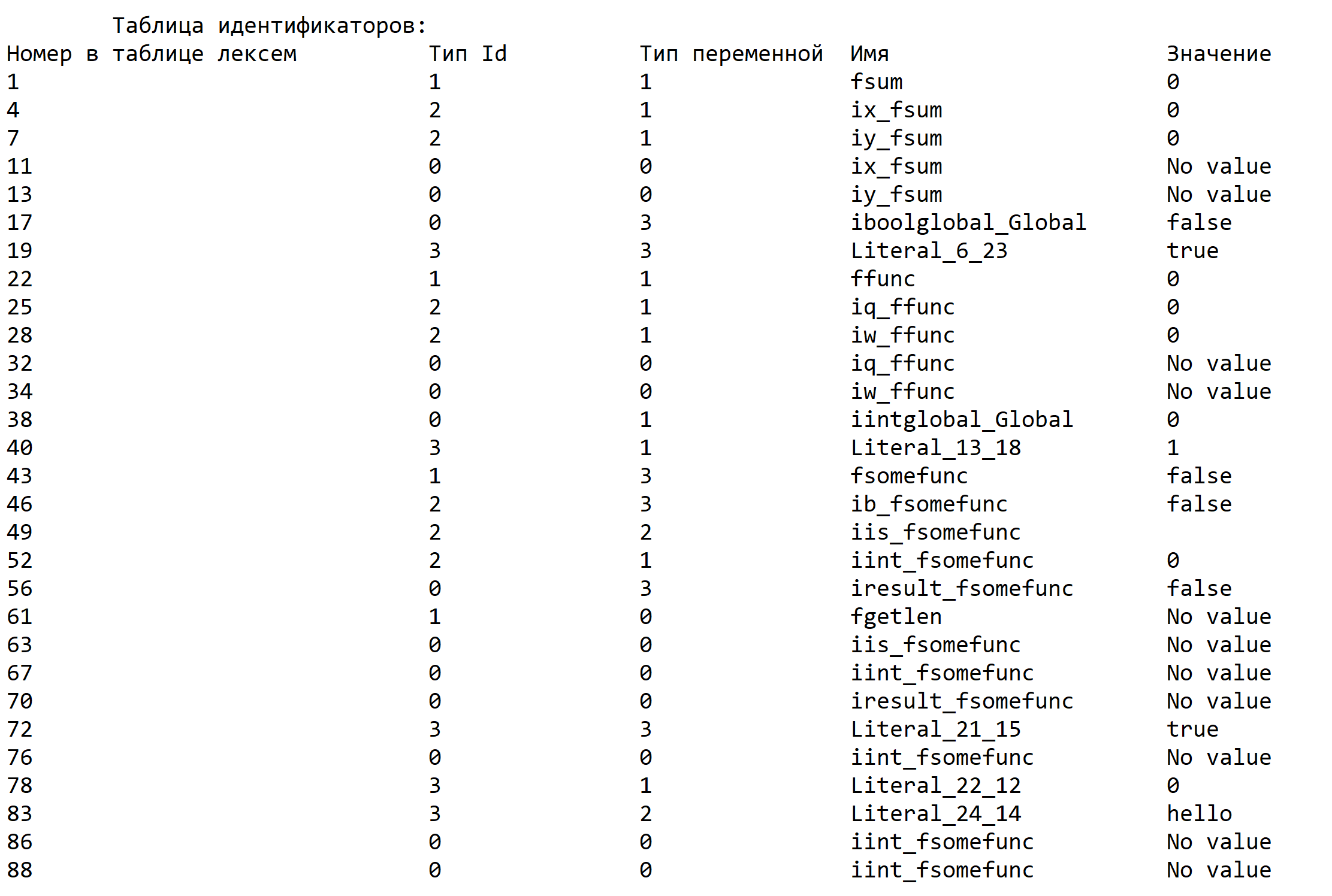
## **Приложения**

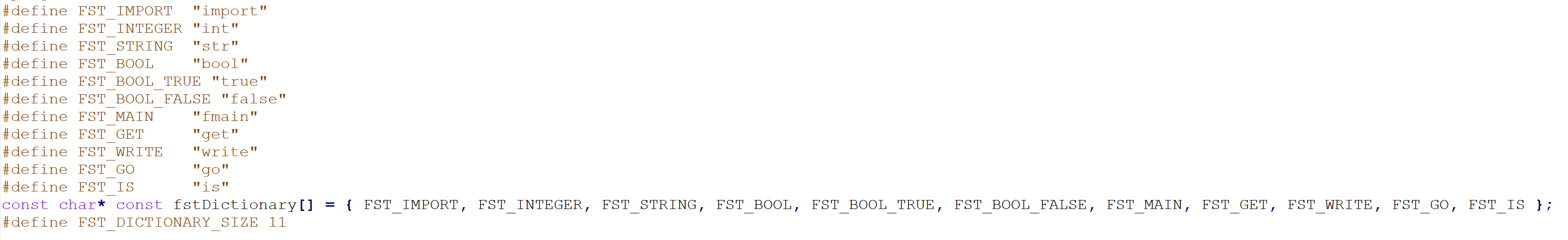
## **Контрольный пример**

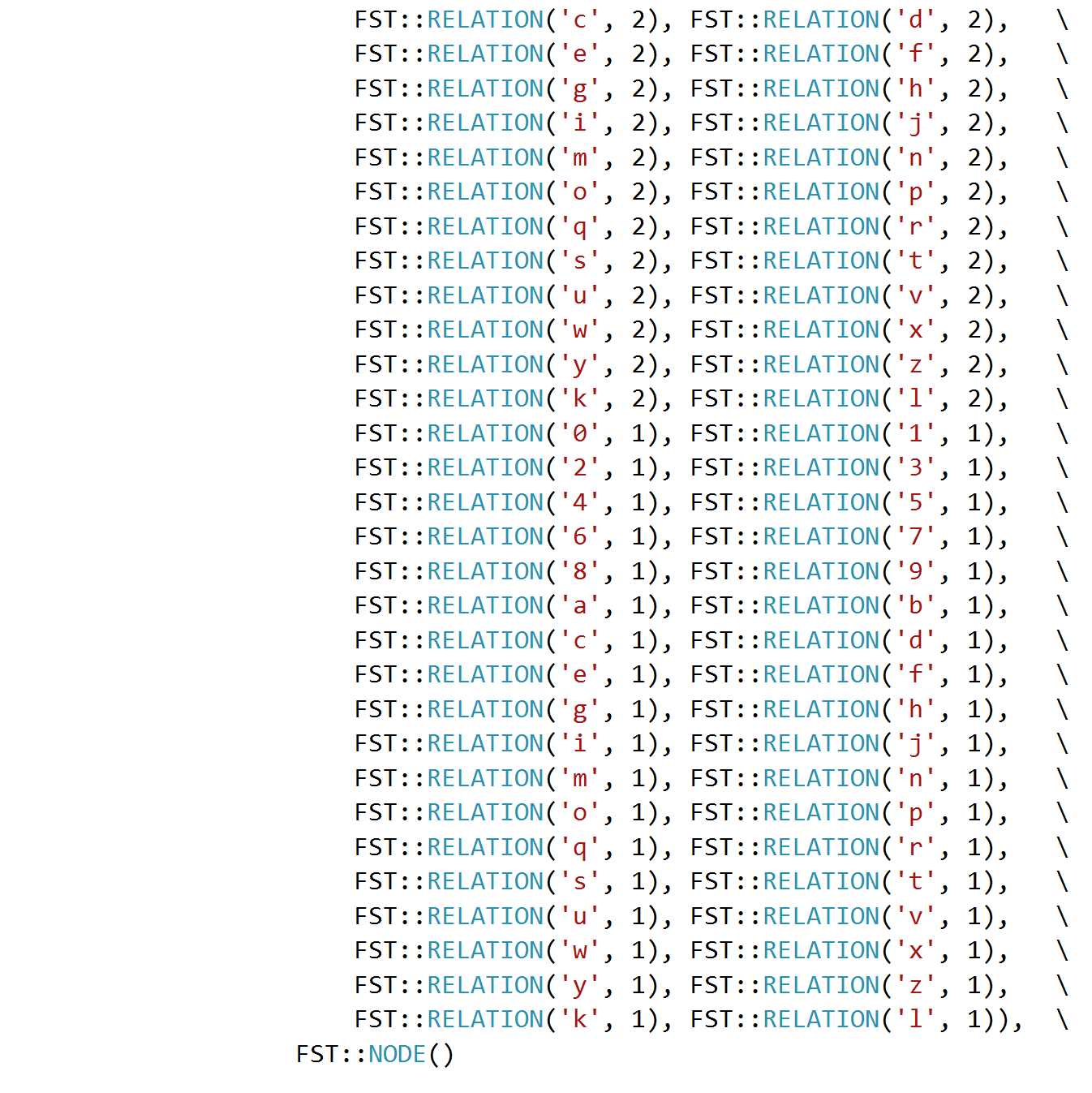
## 

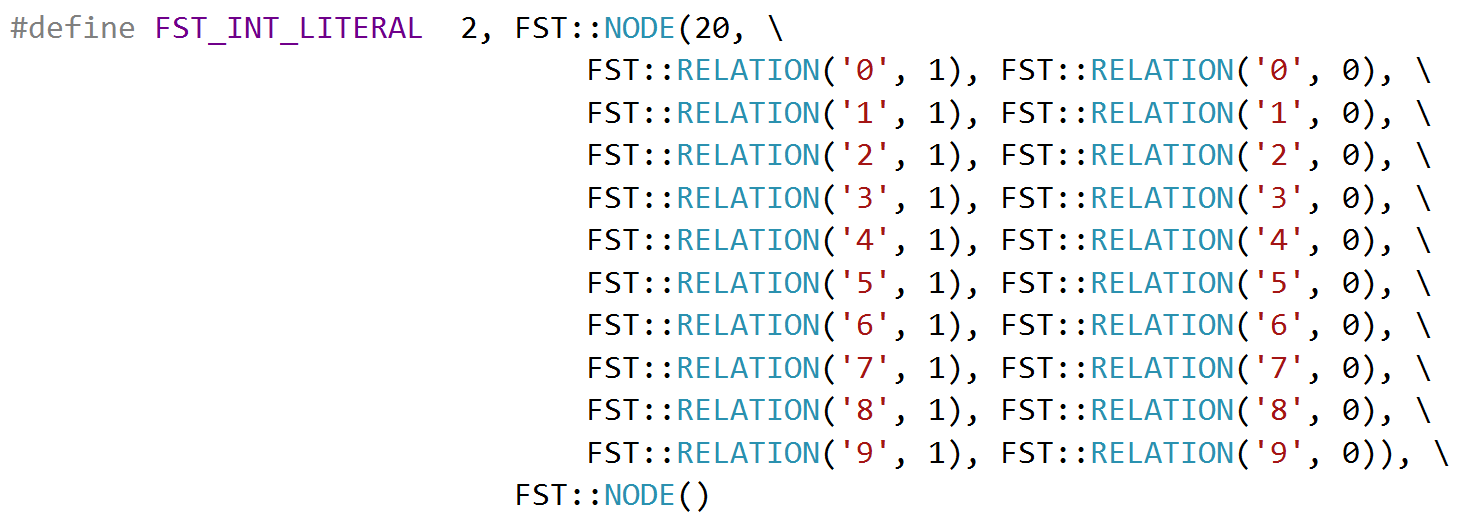
## **Приложение А**

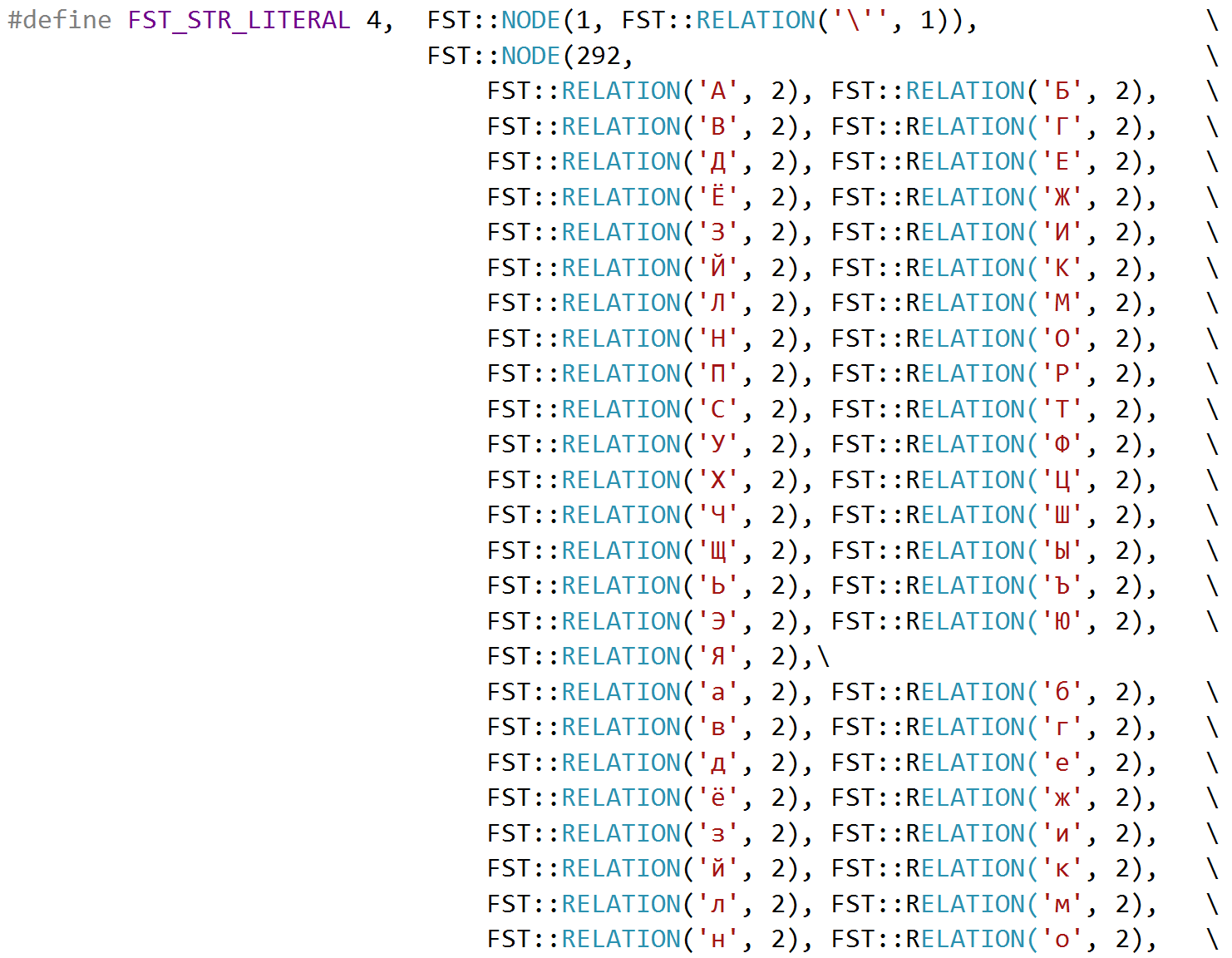


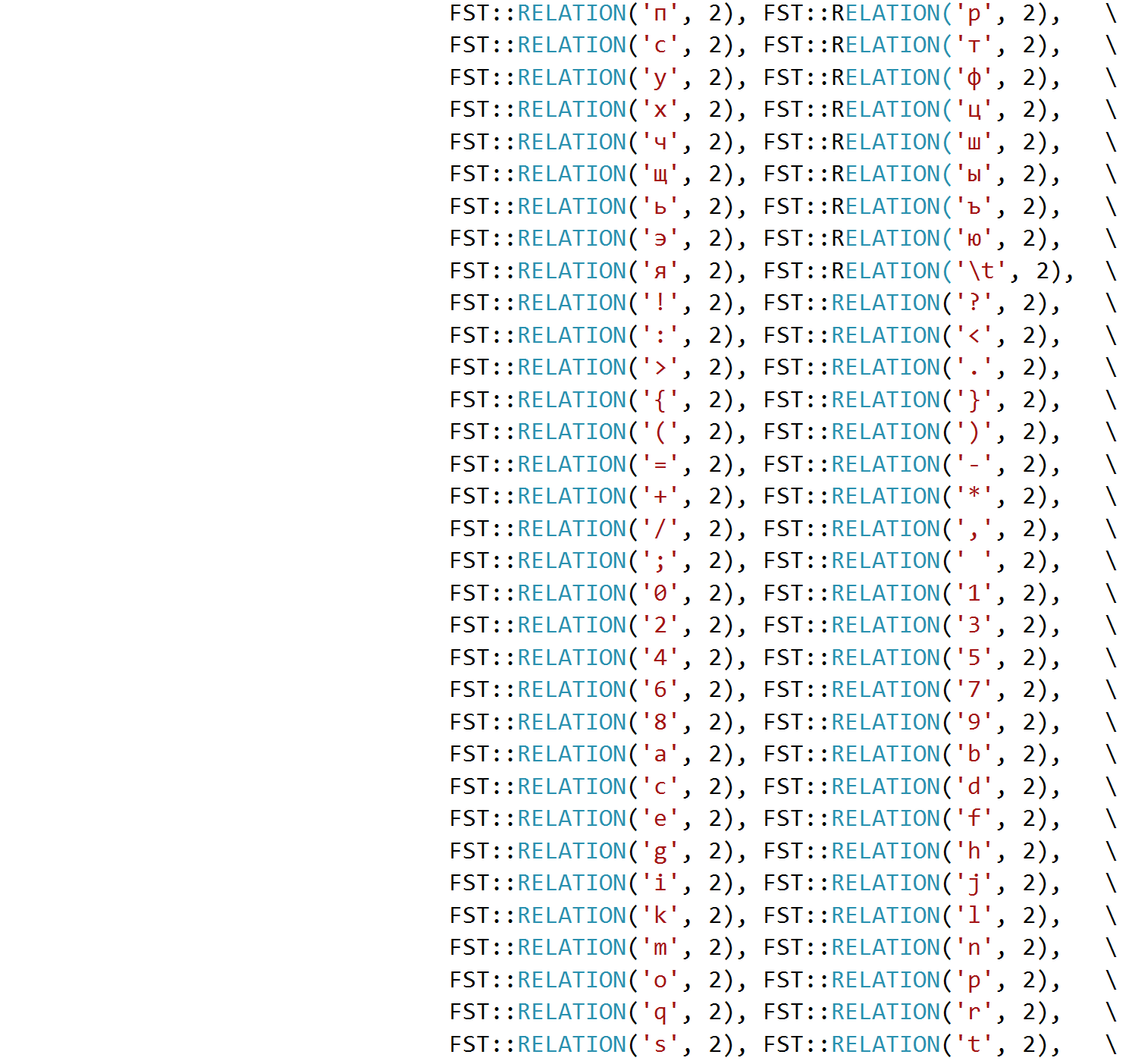


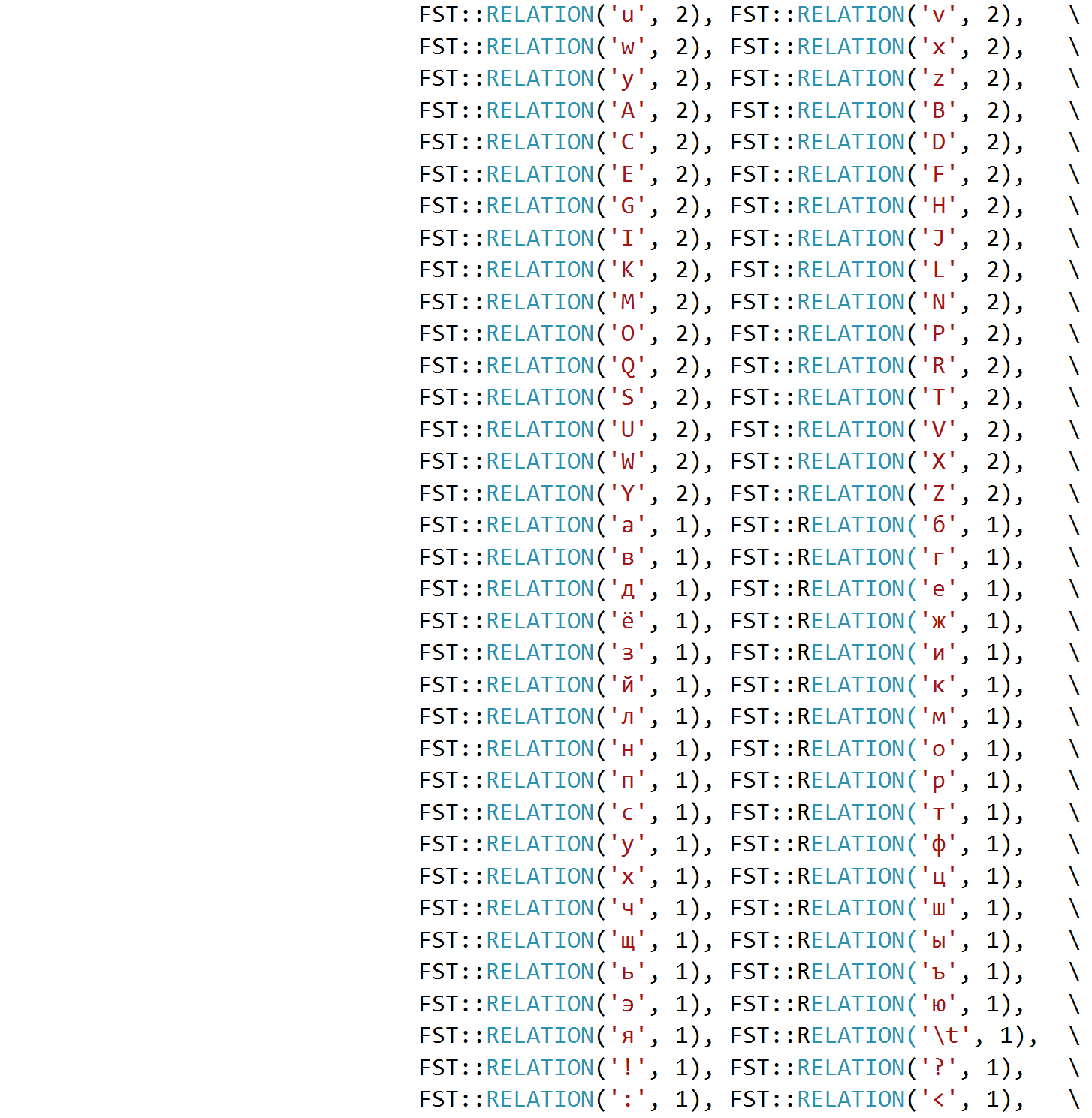


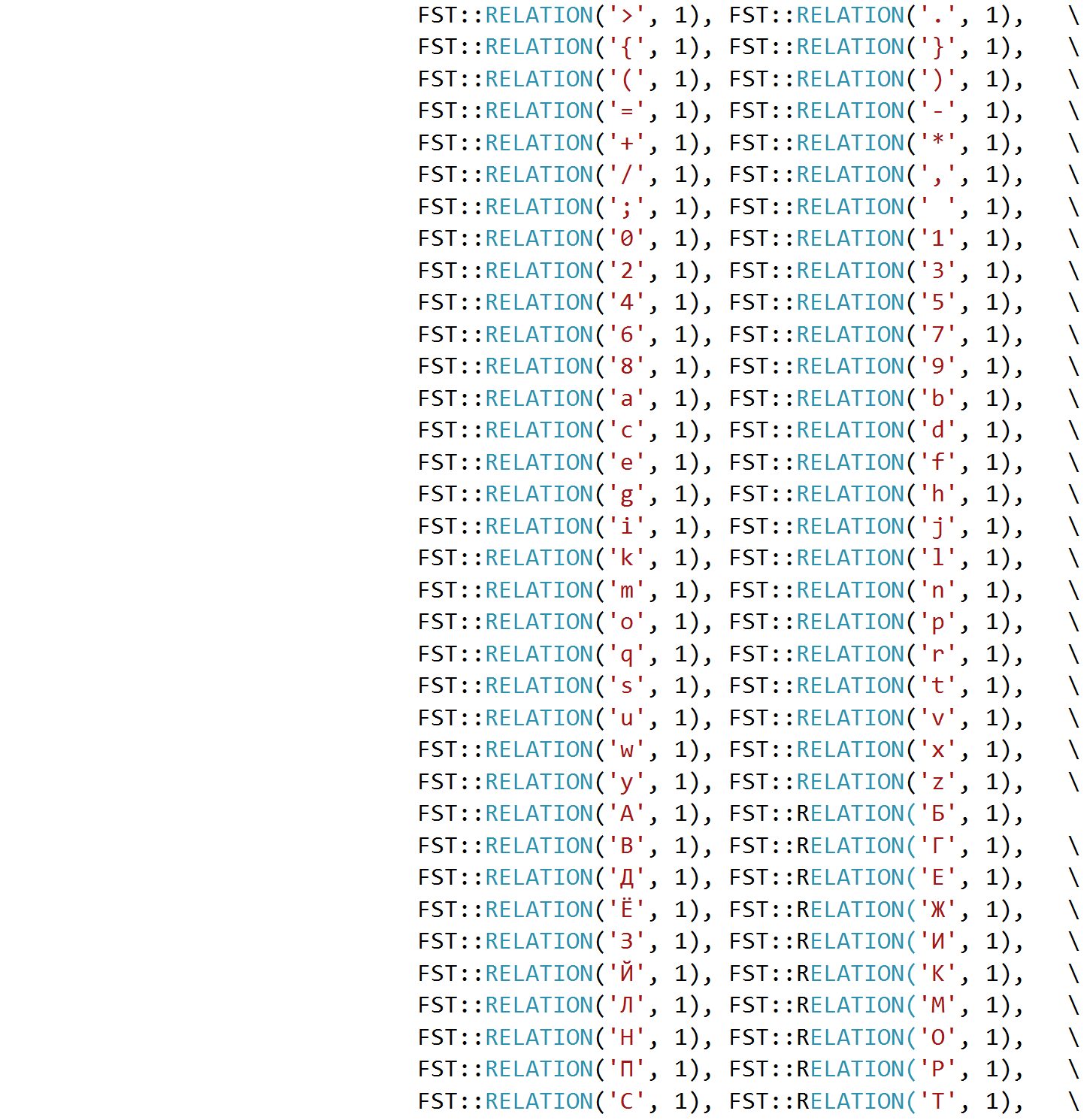


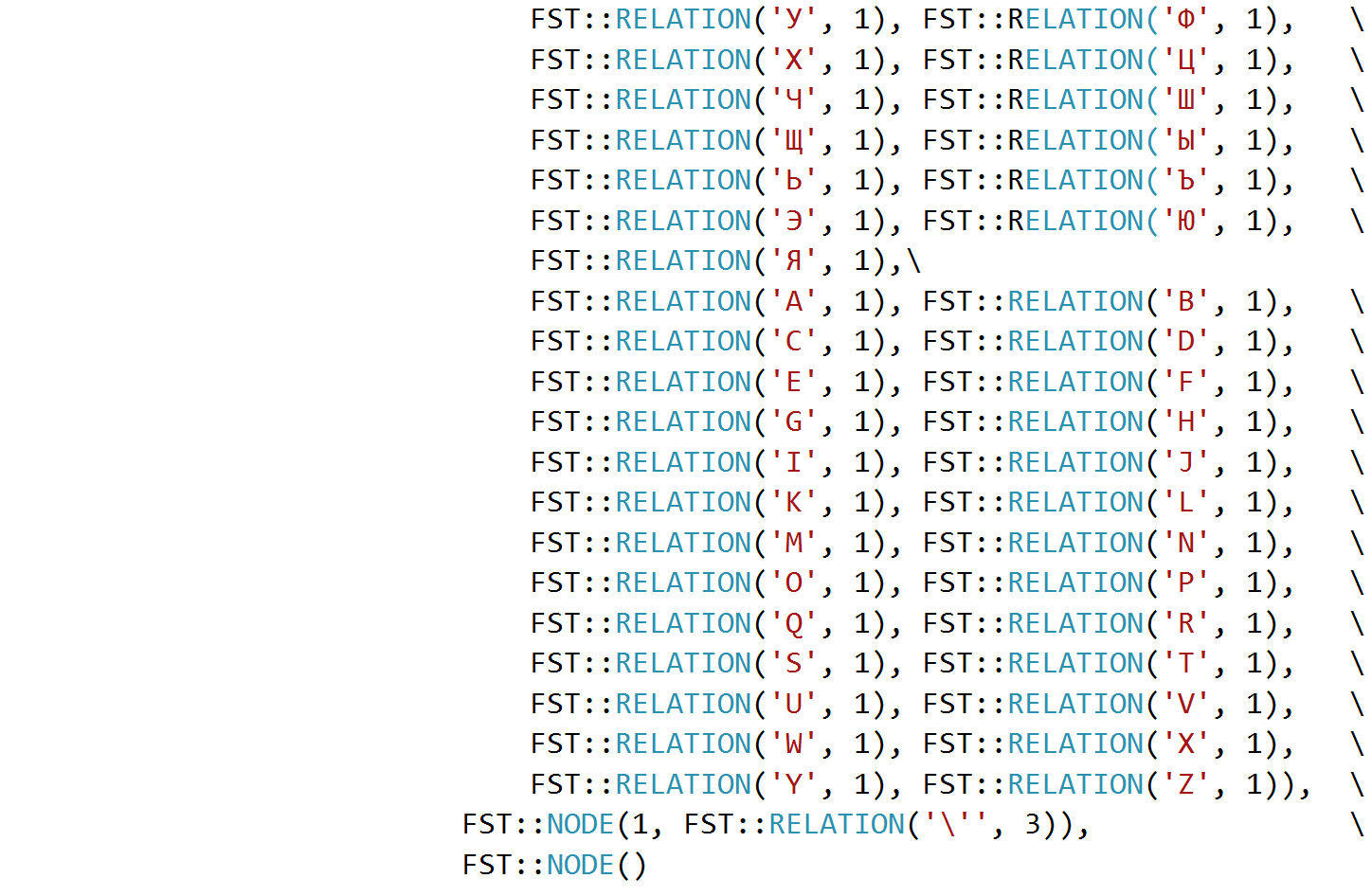


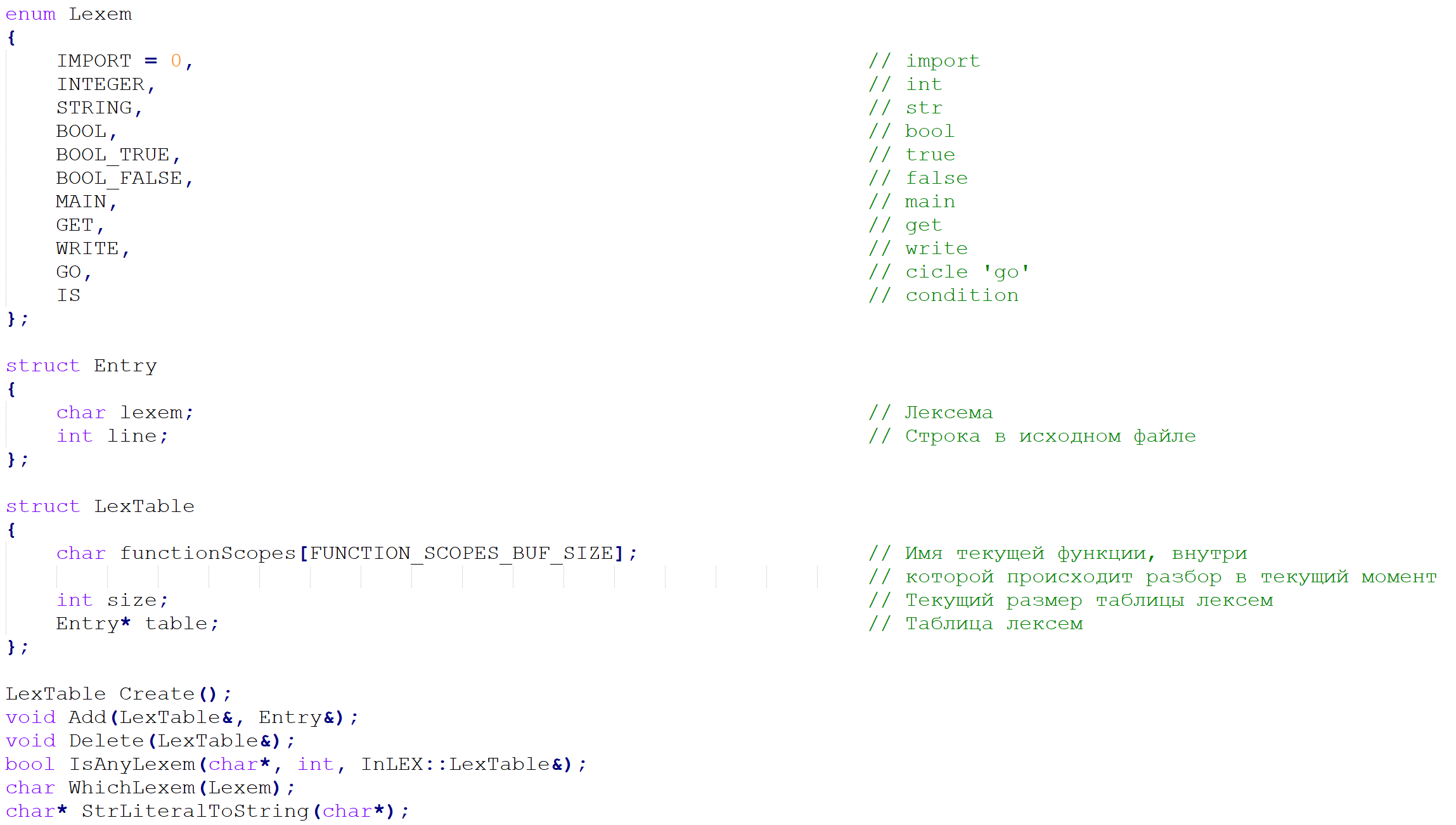


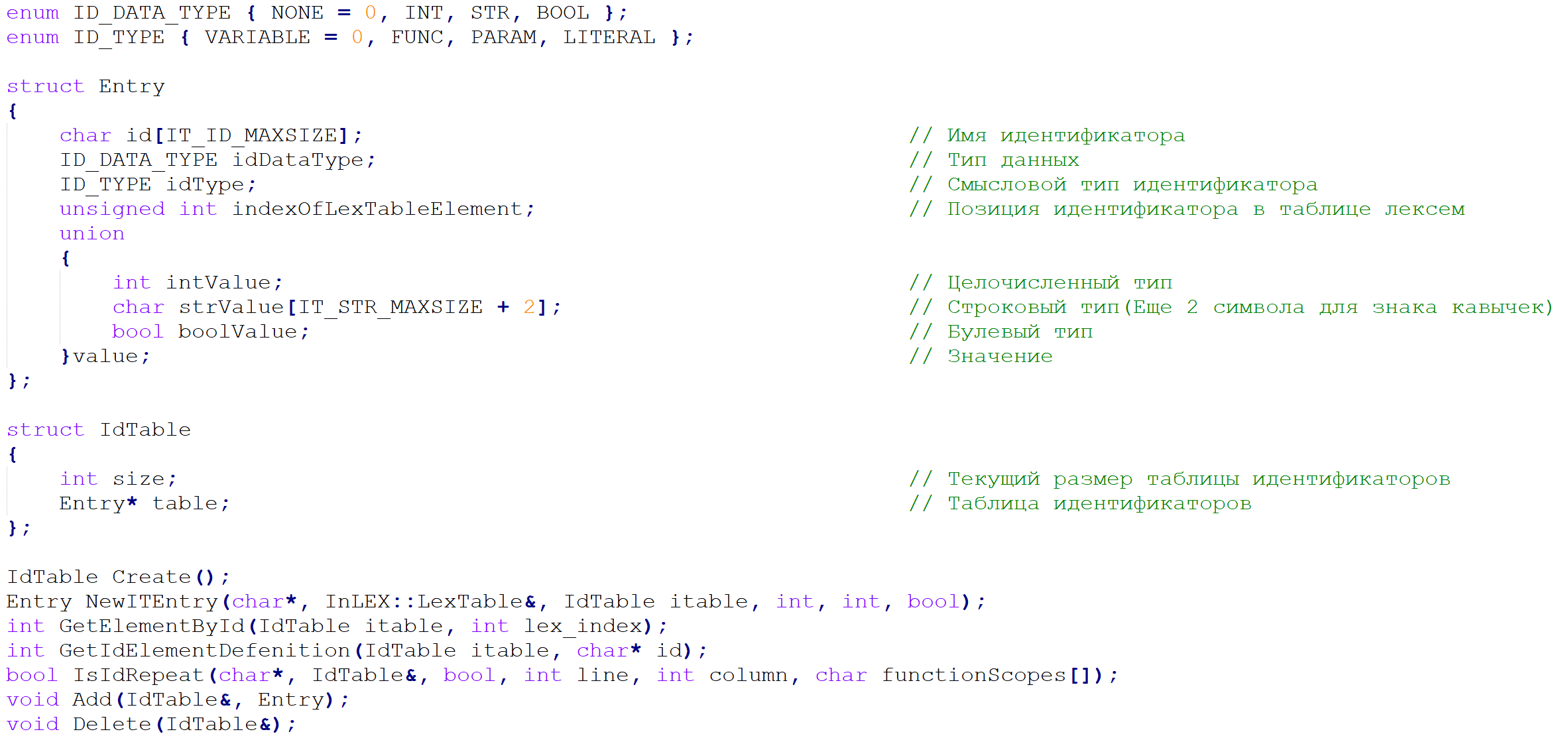




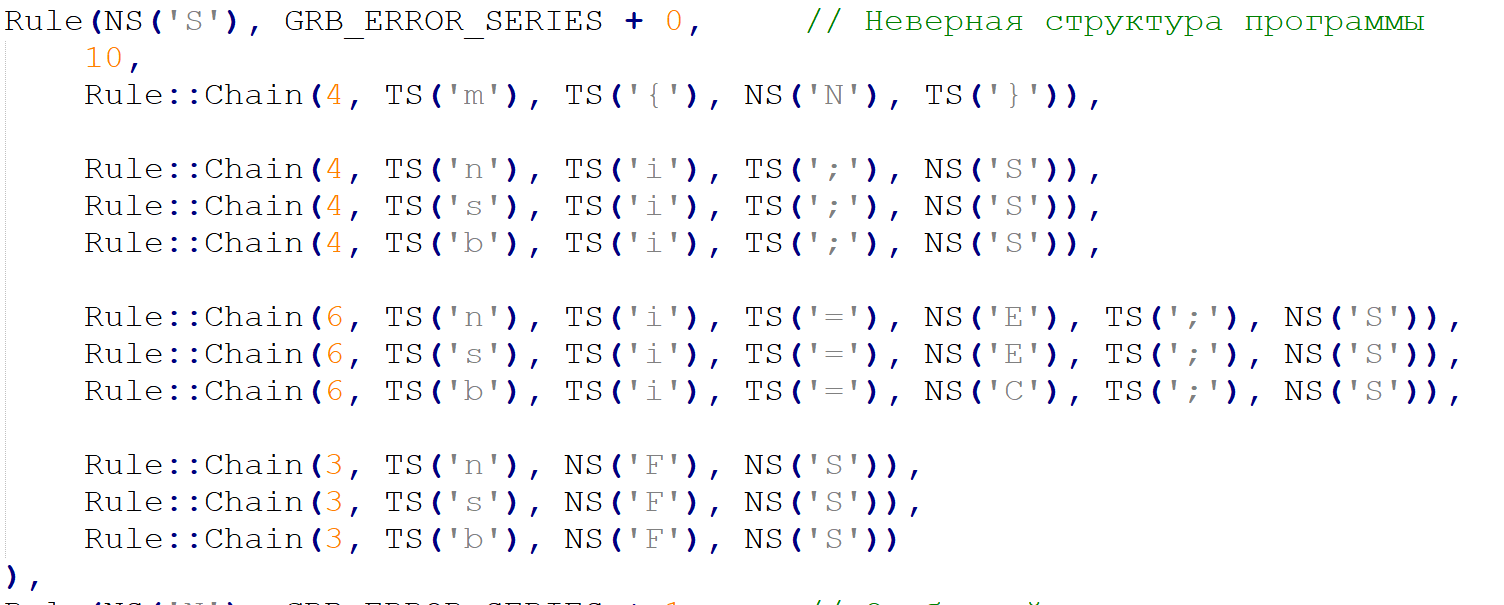


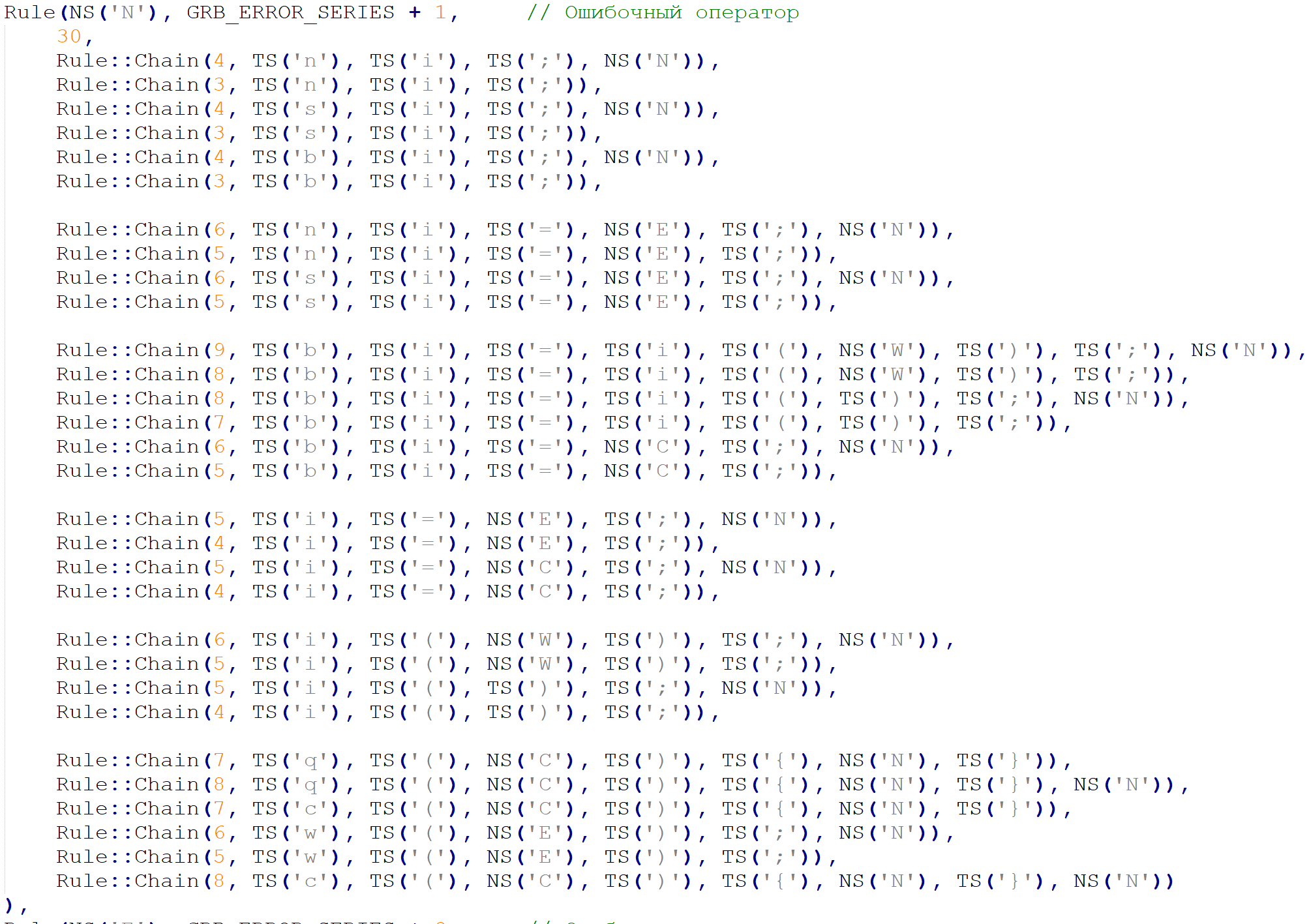


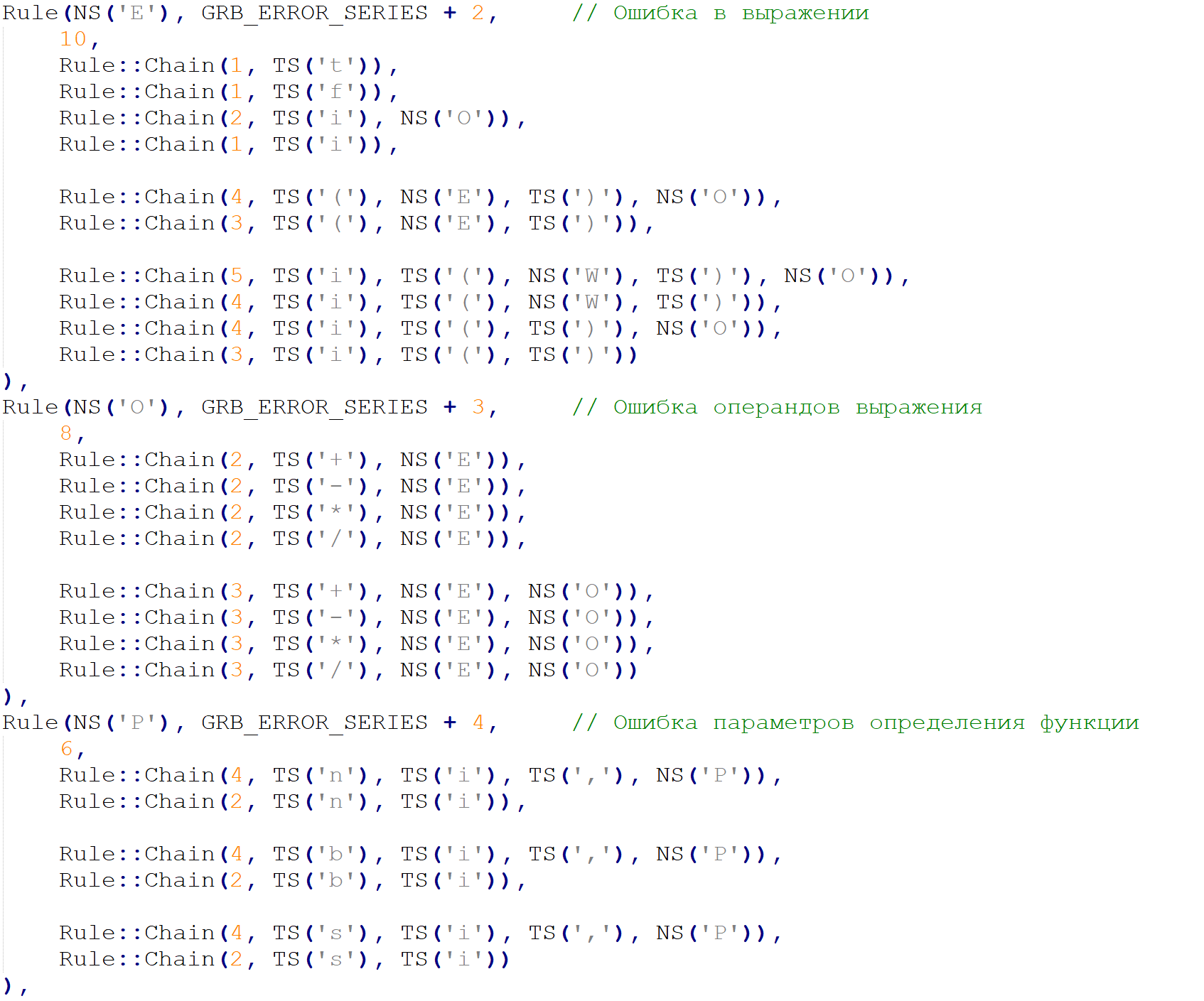


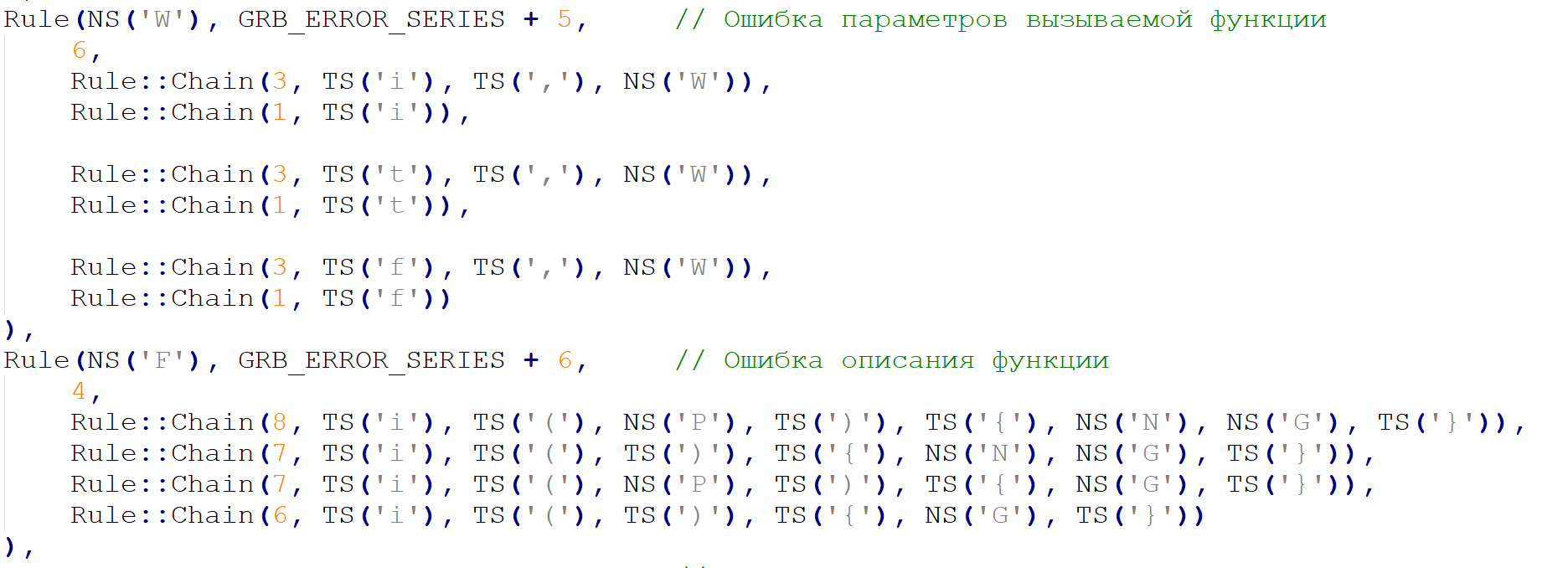


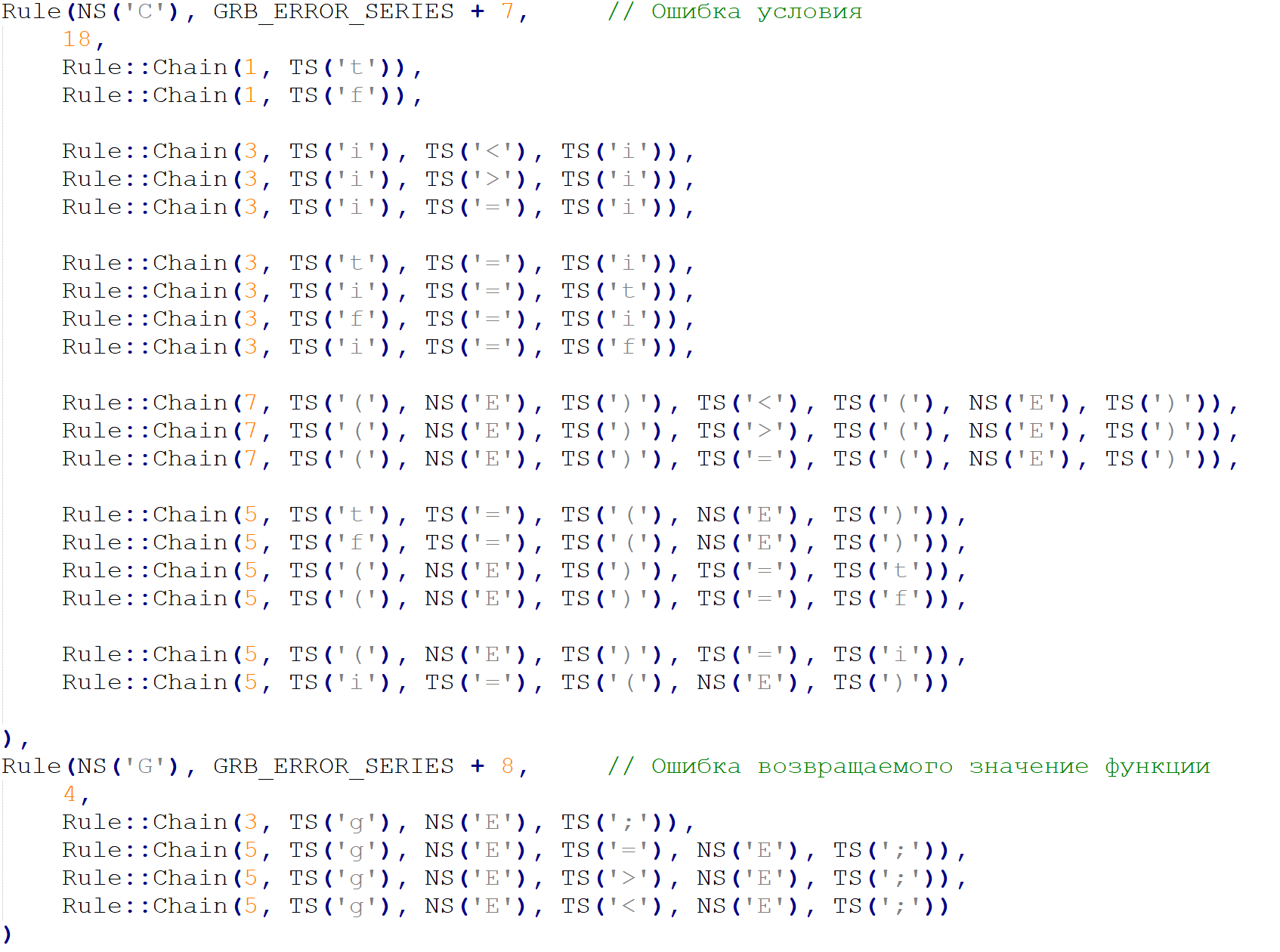
**Приложение Б**





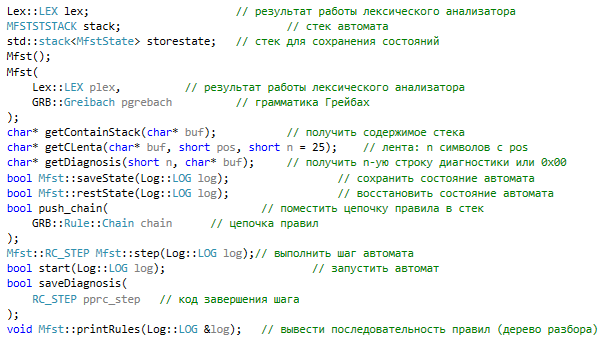






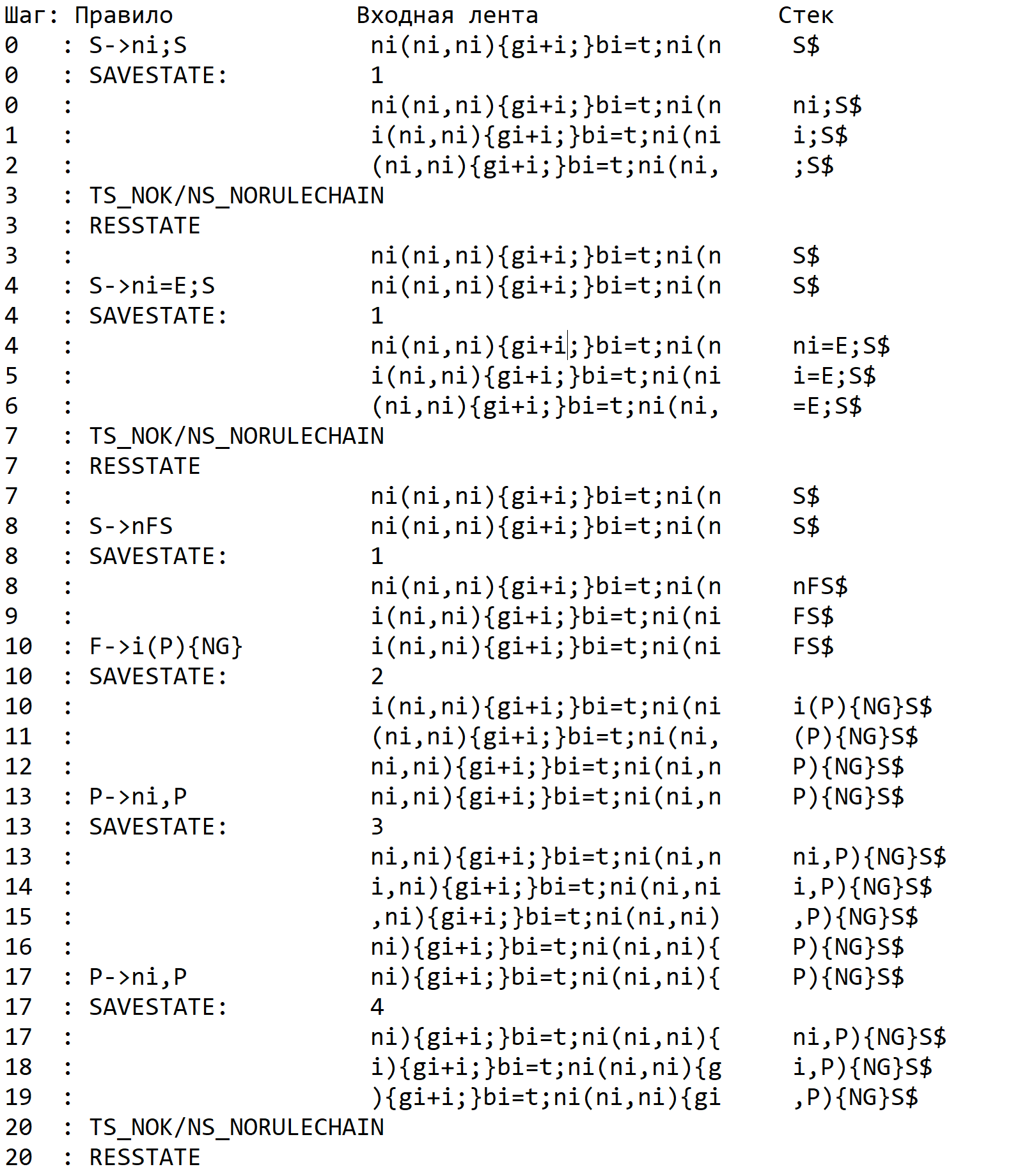
## **Приложение В**



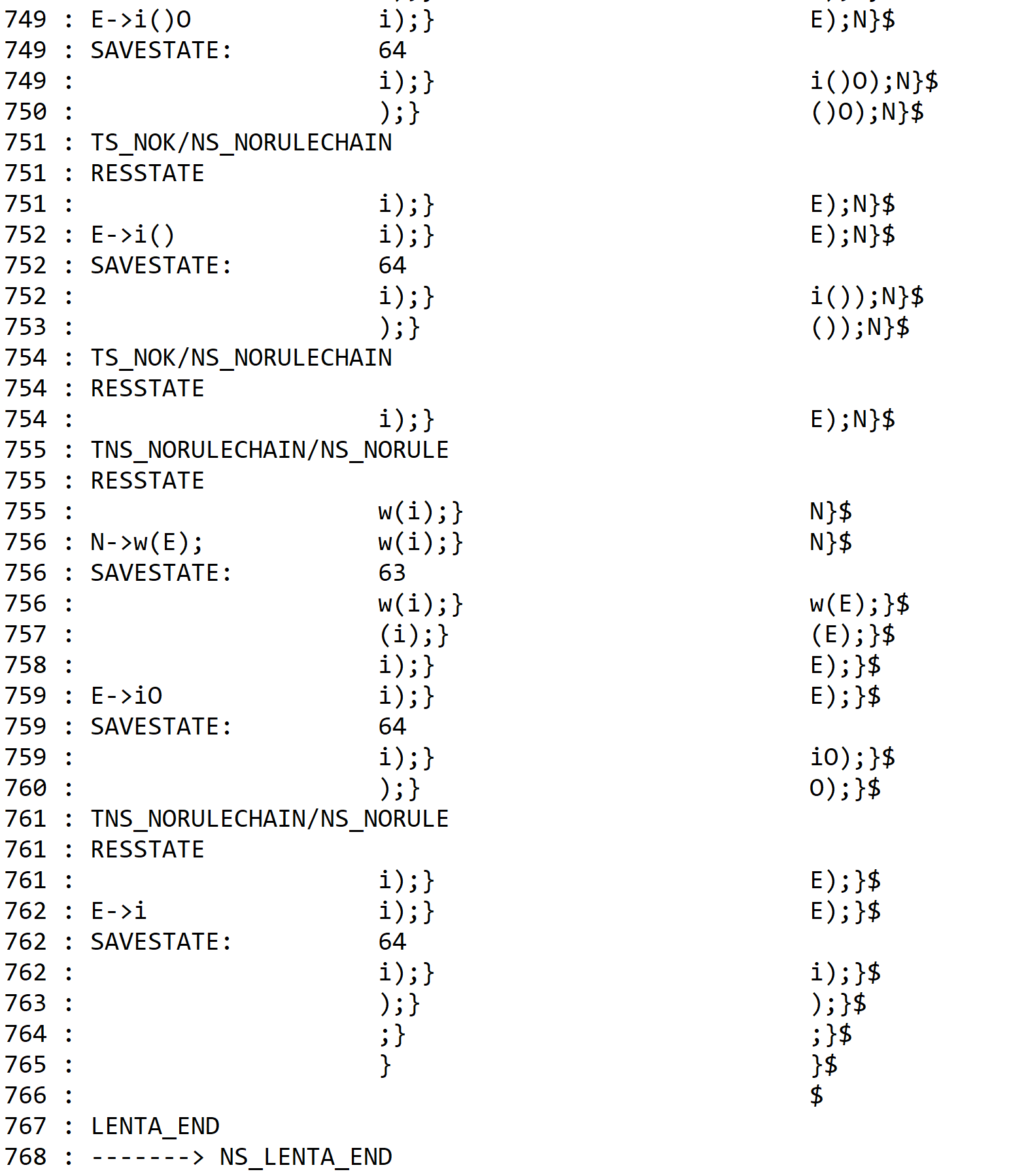


## **Приложение Г**

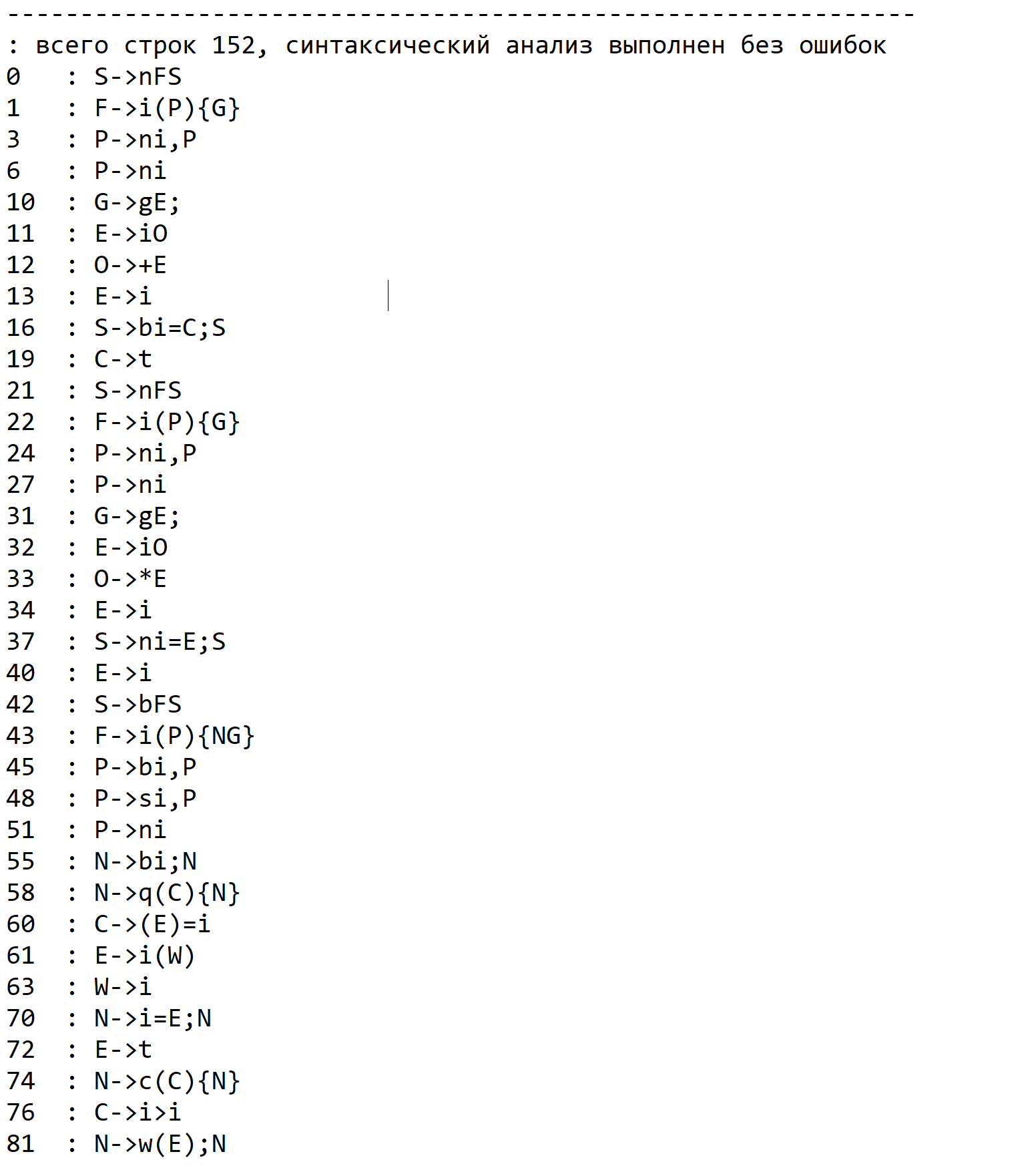
Начало разбора

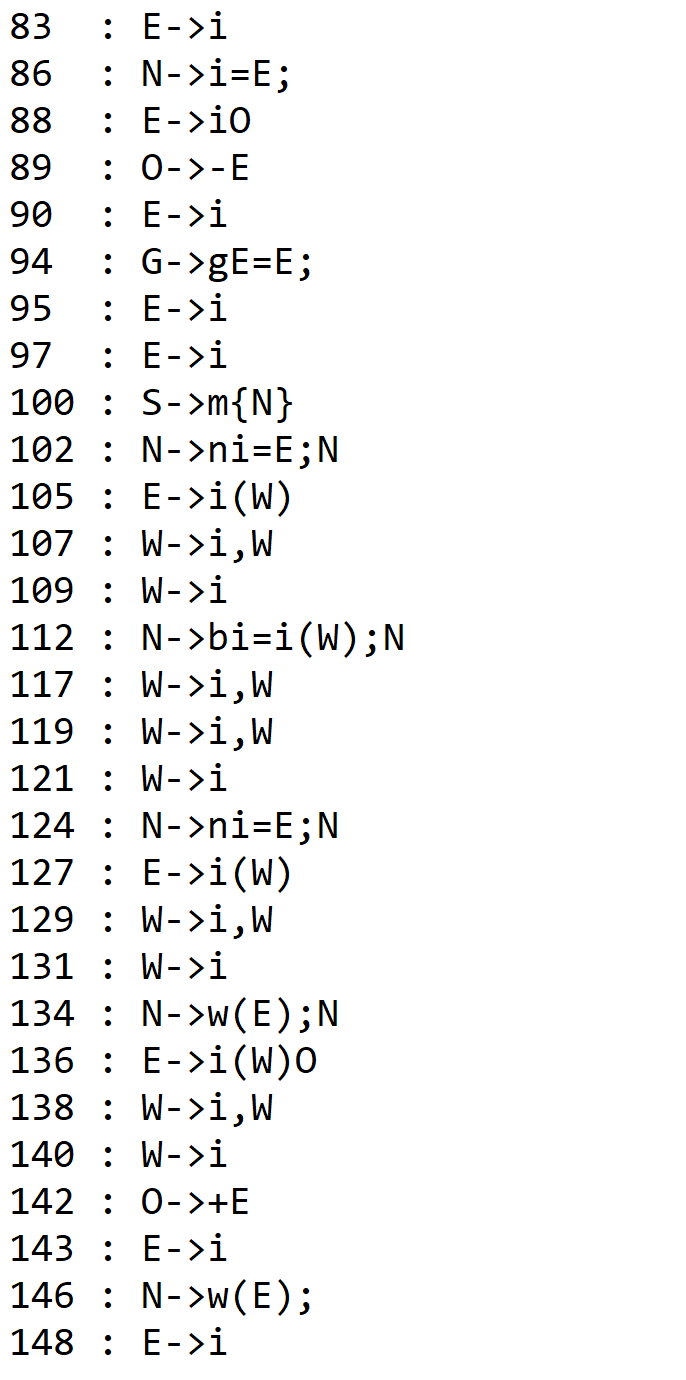


Конец разбора

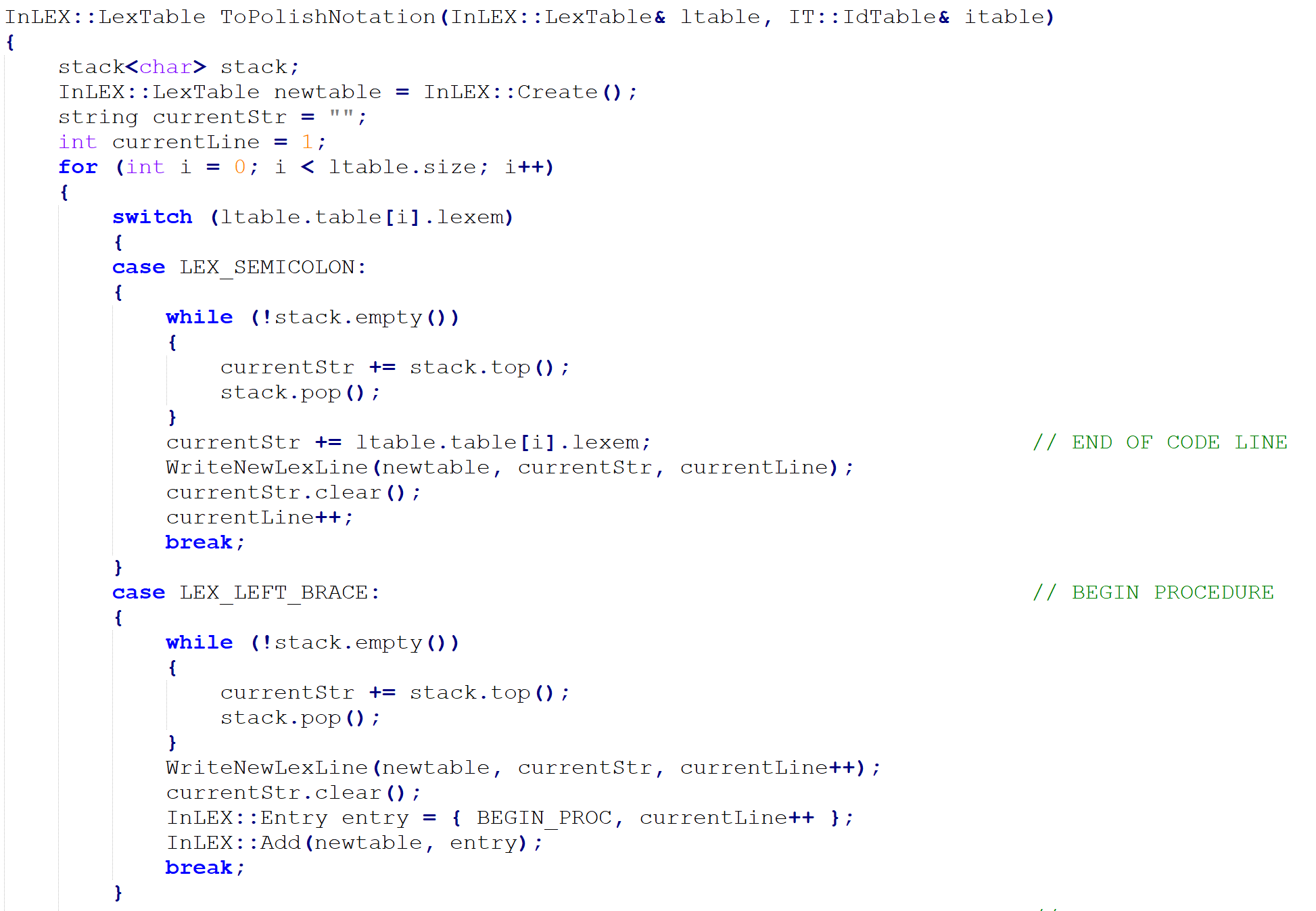


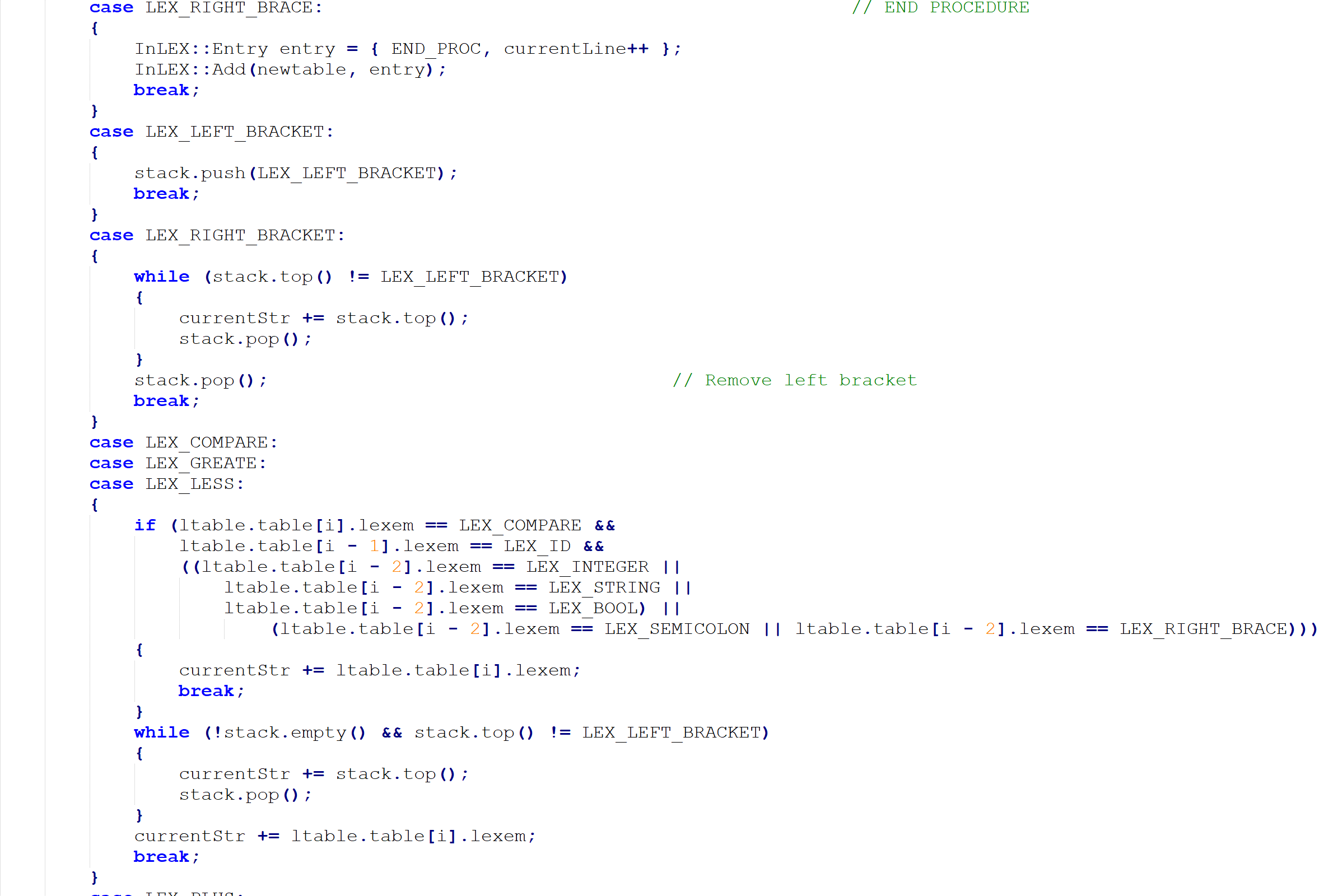
Правила разбора

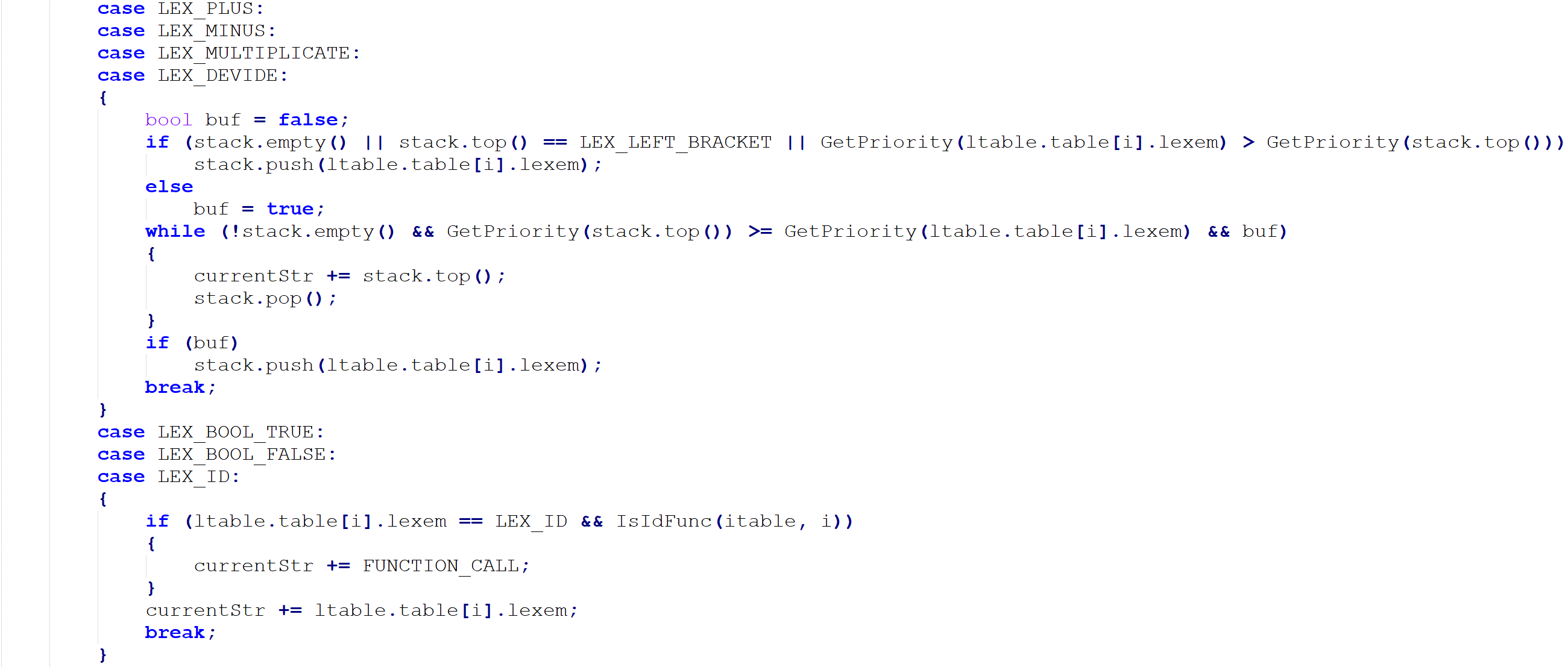


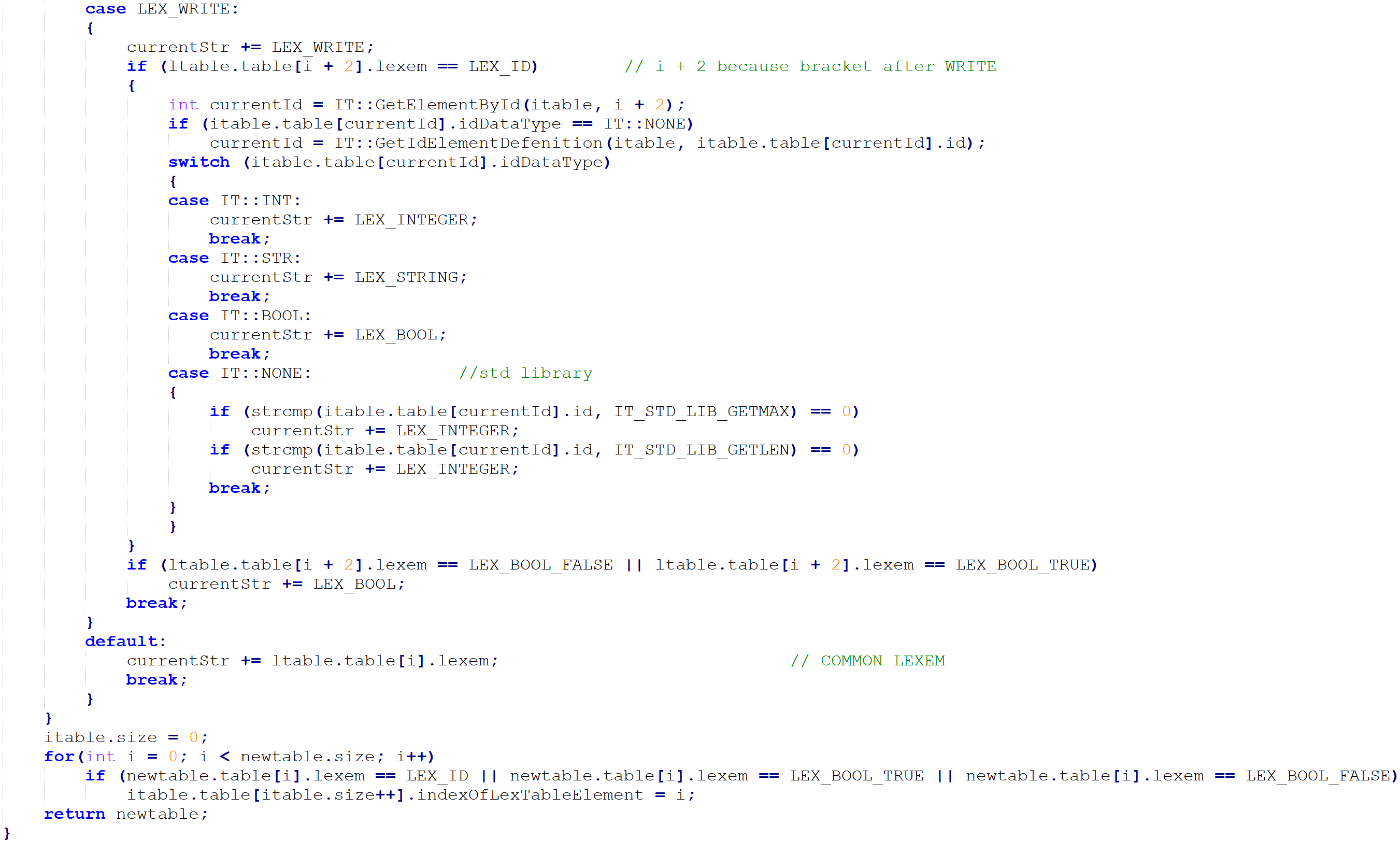


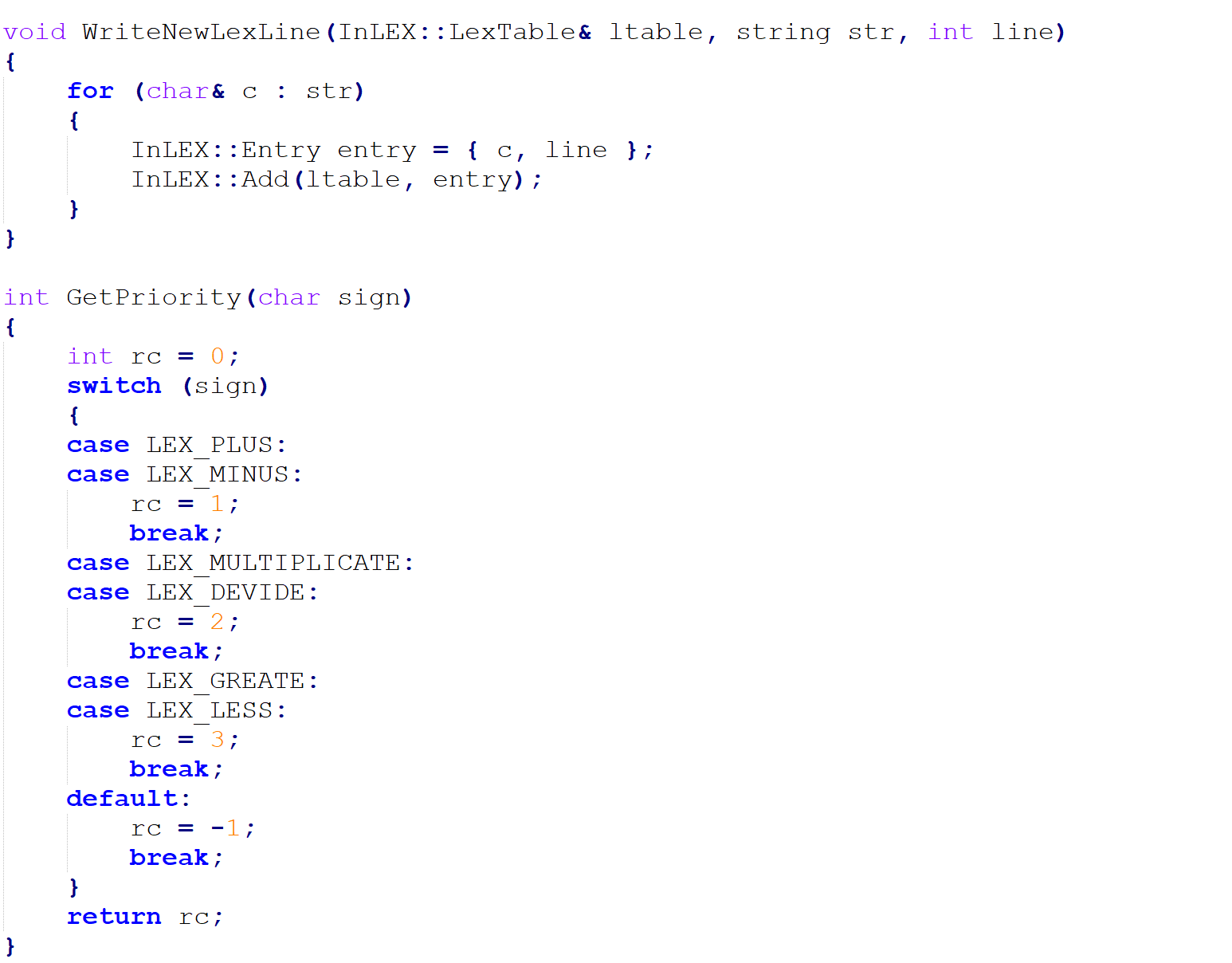
**Приложение Д**

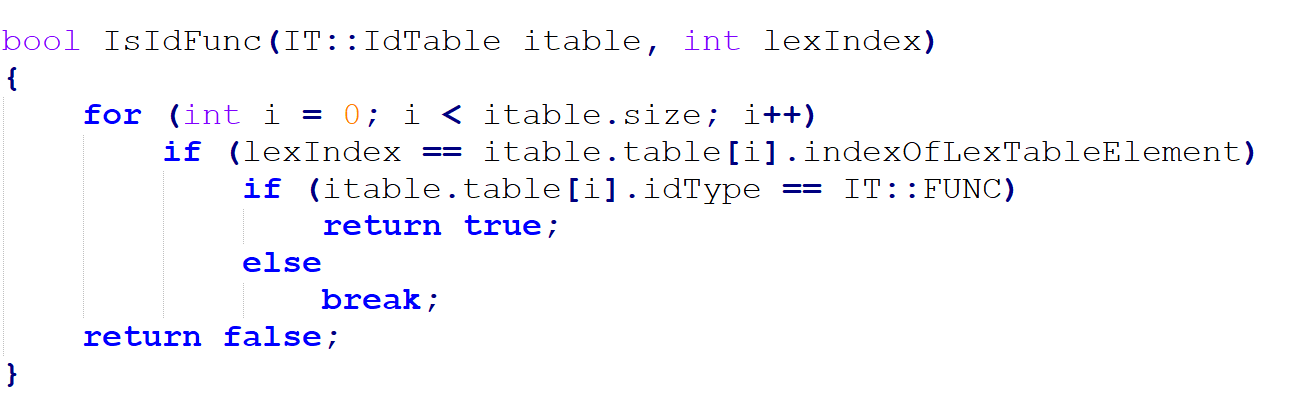


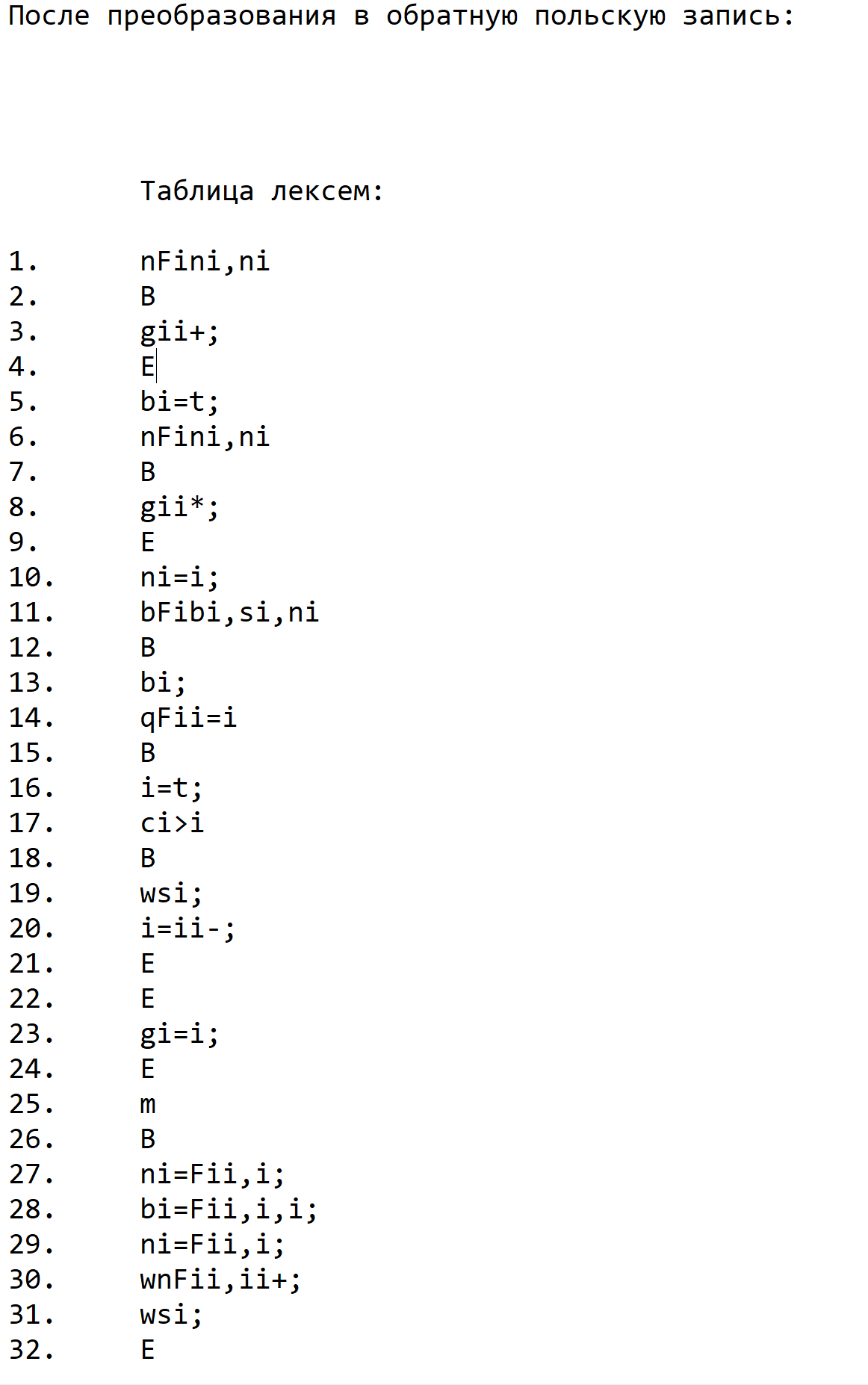


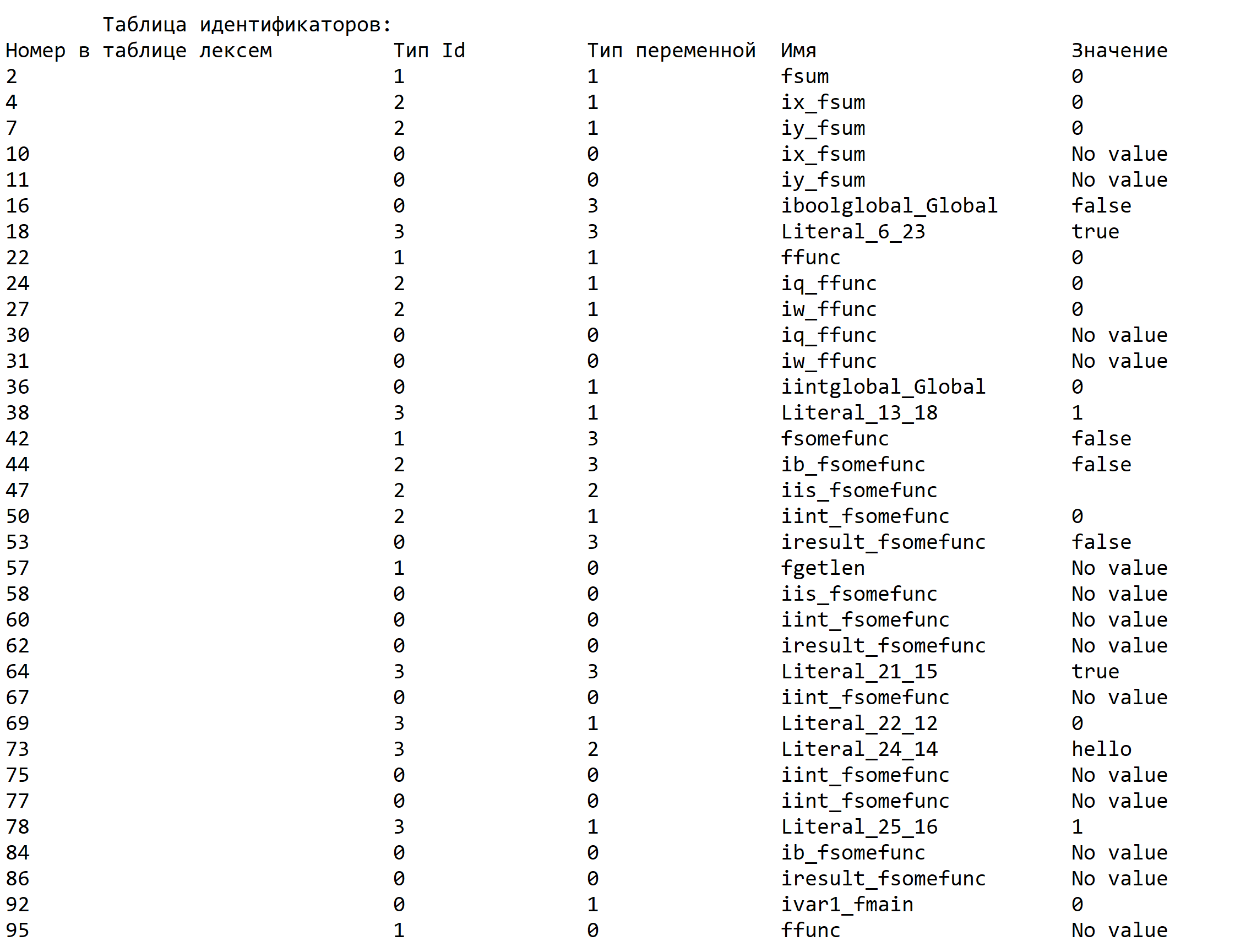


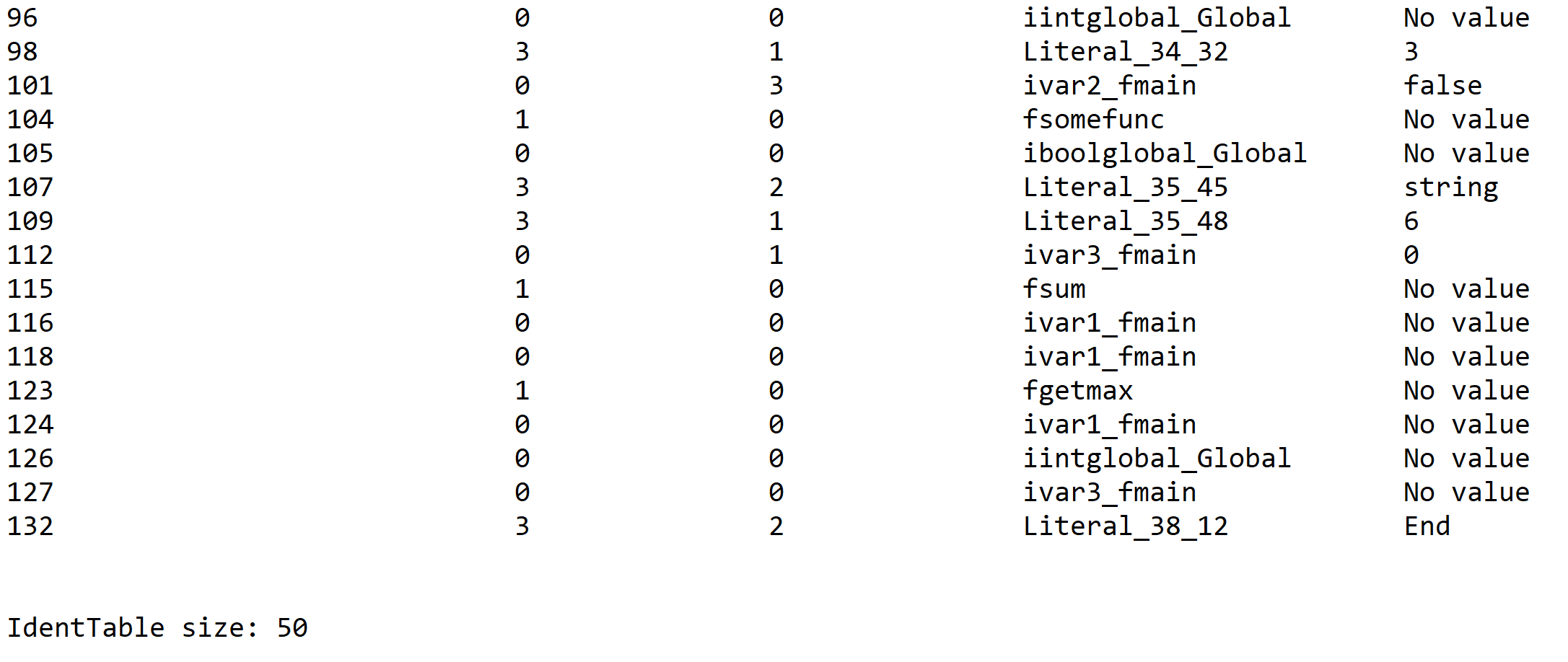












**Приложение Е**

