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 KMO-2022»

Выполнил студент Кришталь Максим Олегович

(Ф.И.О.)

Руководитель проекта преп.-стажёр Карпович М.Н.

(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.-стажёр Карпович М.Н.

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

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

Нормоконтролер преп.-стажёр Карпович М.Н.

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

Курсовой проект защищен с оценкой

Минск 2022

**Оглавление**

[Оглавление 2](#_Toc122402286)

[Введение 5](#_Toc122402287)

[1. Спецификация языка программирования 6](#_Toc122402288)

[1.1 Характеристика языка программирования 6](#_Toc122402289)

[1.2 Определение алфавита языка программирования 6](#_Toc122402290)

[1.3 Применяемые сепараторы 6](#_Toc122402291)

[1.4 Применяемые кодировки 7](#_Toc122402292)

[1.5 Типы данных 7](#_Toc122402293)

[1.6 Преобразование типов данных 7](#_Toc122402294)

[1.7 Идентификаторы 7](#_Toc122402295)

[1.8 Литералы 8](#_Toc122402296)

[1.9 Объявление данных 8](#_Toc122402297)

[1.10 Инициализация данных 9](#_Toc122402298)

[1.11 Инструкции языка 9](#_Toc122402299)

[1.12 Операции языка 10](#_Toc122402300)

[1.13 Выражения и их вычисления 10](#_Toc122402301)

[1.14 Конструкции языка 10](#_Toc122402302)

[1.15 Область видимости идентификаторов 11](#_Toc122402303)

[1.16 Семантические проверки 11](#_Toc122402304)

[1.17 Распределение оперативной памяти на этапе выполнения 12](#_Toc122402305)

[1.18 Стандартная библиотека и ее состав 12](#_Toc122402306)

[1.19 Ввод и вывод данных 12](#_Toc122402307)

[1.20 Точка входа 13](#_Toc122402308)

[1.21 Препроцессор 13](#_Toc122402309)

[1.22 Соглашения о вызове 13](#_Toc122402310)

[1.23 Объектный код 13](#_Toc122402311)

[1.24 Классификация сообщений транслятора 13](#_Toc122402312)

[1.25 Контрольный пример 13](#_Toc122402313)

[2 Структура транслятора 14](#_Toc122402314)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 14](#_Toc122402315)

[2.2 Перечень входных параметров транслятора 15](#_Toc122402316)

[2.3 Перечень протоколов, формируемых транслятором и их содержимое 15](#_Toc122402317)

[3 Разработка лексического анализатора 16](#_Toc122402318)

[3.1 Структура лексического анализатора 16](#_Toc122402319)

[3.2 Контроль входных символов 16](#_Toc122402320)

[3.3 Удаление избыточных символов 17](#_Toc122402321)

[3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем 17](#_Toc122402322)

[3.5 Основные структуры данных 18](#_Toc122402323)

[3.6 Структура и перечень сообщений лексического анализатора 18](#_Toc122402324)

[3.7 Принцип обработки ошибок 18](#_Toc122402325)

[3.8 Параметры лексического анализатора 18](#_Toc122402326)

[3.9 Алгоритм лексического анализа 18](#_Toc122402327)

[3.10 Контрольный пример 18](#_Toc122402328)

[4 Разработка синтаксического анализатора 19](#_Toc122402329)

[4.1 Структура синтаксического анализатора 19](#_Toc122402330)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 19](#_Toc122402331)

[4.3 Построение конечного магазинного автомата 20](#_Toc122402332)

[4.4 Основные структуры данных 20](#_Toc122402333)

[4.5 Описание алгоритма синтаксического разбора 21](#_Toc122402334)

[4.6 Структура и перечень сообщений синтаксического анализатора 21](#_Toc122402335)

[4.7 Параметры синтаксического анализатора и режимы его работы 21](#_Toc122402336)

[4.8 Принцип обработки ошибок 22](#_Toc122402337)

[4.9 Контрольный пример 22](#_Toc122402338)

[5 Разработка семантического анализатора 23](#_Toc122402339)

[5.1 Структура семантического анализатора 23](#_Toc122402340)

[5.2 Функции семантического анализатора 23](#_Toc122402341)

[5.3 Структура и перечень сообщений семантического анализатора 23](#_Toc122402342)

[5.4 Принцип обработки ошибок 24](#_Toc122402343)

[5.5 Контрольный пример 24](#_Toc122402344)

[6 Преобразование выражений 25](#_Toc122402345)

[6.1. Выражения, допускаемые языком 25](#_Toc122402346)

[6.2 Польская запись и принцип ее построения 25](#_Toc122402347)

[6.3 Программная реализация обработки выражений 26](#_Toc122402348)

[6.4 Контрольный пример 26](#_Toc122402349)

[7 Генерация кода 27](#_Toc122402350)

[7.1 Структура генератора кода 27](#_Toc122402351)

[7.2 Представление типов данных в оперативной памяти 27](#_Toc122402352)

[7.3 Статическая библиотека 27](#_Toc122402353)

[7.4 Входные параметры генератора кода 28](#_Toc122402354)

[7.5 Контрольный пример 28](#_Toc122402355)

[8 Тестирование транслятора 29](#_Toc122402356)

[8.1 Тестирование лексического анализатора 29](#_Toc122402357)

[8.2 Тестирование синтаксического анализатора 29](#_Toc122402358)

[8.3 Тестирование семантического анализатора 29](#_Toc122402359)

[Заключение 31](#_Toc122402360)

[Список использованных источников 32](#_Toc122402361)

[Приложение А 33](#_Toc122402362)

[Приложение В 35](#_Toc122402363)

[Приложение Г 36](#_Toc122402364)

[Приложение Д 38](#_Toc122402365)

[Приложение Е 40](#_Toc122402366)

[Приложение Ж 42](#_Toc122402367)

[Приложение З 45](#_Toc122402368)

[Приложение И 47](#_Toc122402369)

# **Введение**

Основной целью данной курсовой работы является разработка транслятора для языка программирования KMO-2022. Главная задача транслятора заключается в том, чтобы исходный код на языке KMO-2022 в код на языке Assembler.

Исходя из цели курсового проекта, были определены следующие задачи:

1. Разработка спецификации языка программирования;
2. Разработка структуры транслятора;
3. Разработка лексического анализатора;
4. Разработка синтаксического анализатора;
5. Разработка семантического анализатора;
6. Обработка выражений;
7. Генерация кода на язык Assembler;
8. Тестирование транслятора.

# **1. Спецификация языка программирования**

## 1.1 Характеристика языка программирования

KMO-2022 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 KMO-2022 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, приведены в таблице 1.1.

Таблица 1.1 Символы-сепараторы

|  |  |
| --- | --- |
| Символы | Назначение |
| (…) | Повышение приоритета арифметических операций, ограничение блока параметров функции. |
| <…> | Ограничение тела функции или цикла. |
| ; | Разделитель программных конструкций. |
| + - \* / % | Арифметические операции. |
| = | Оператор присваивания. |
| , | Разделитель параметров функции. |
| $ | Строка комментария. |
| «пробел» | Разделитель цепочек языка, не допустим и именах идентификаторов и ключевых словах языка. |

## 1.4 Применяемые кодировки

Для написания программ на языке KMO-2022 применяется кодировка Windows-1251.

## 1.5 Типы данных

В языке KMO-2022 реализованы 2 фундаментальных типа данных: целочисленный и строковый, которые описываются в в таблице 1.2.

Таблица 1.2 Типы данных языка KMO-2022

|  |  |
| --- | --- |
| 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. |
| 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 KMO-2022 не поддерживается.

## 1.7 Идентификаторы

Общее число идентификаторов не должно превышать максимальный размер таблицы идентификаторов (4096). Длина идентификатора не должна превышать 5 символов. В именах идентификаторов допустимо использование символов нижнего регистров и цифр. Идентификаторы не должны совпадать с ключевыми словами языка и не должны начинаться с цифр. К типам идентификаторов относятся: имя переменной, имя функции, имя параметра.

## 1.8 Литералы

В языке KMO-2022 предусмотрены строковые и целочисленные литералы. Язык поддерживает представление целочисленных литералов в шестнадцатеричной, десятичной, восьмеричной и двоичной системе счисления. Их описание представлено в таблице 1.3.

Таблица 1.3 Характеристика литералов языка KMO-2022

|  |  |
| --- | --- |
| Литералы | Характеристика |
| Строковые | Набор символов алфавита языка, заключенный в одинарные кавычки (‘’). Допустимо использование символов только нижнего регистра, цифр и некоторых специальных символов. |
| Целочисленные в двоичной форме | Последовательность цифр 0 и 1 с обязательным префиксом “Bx”: Bx011010. |
| Целочисленные в восьмеричной форме | Последовательность цифр 0…7 с обязательным префиксом “Qx”: Qx75. |
| Целочисленные в десятичной форме | Последовательность цифр 0…9 без префиксов: 255. |
| Целочисленные в шестнадцатеричной форме | Последовательность цифр 0…9 и символов A…F с обязательным префиксом “Hx”: HxA5. |
| Отрицательные целочисленные | Для того, чтобы получить отрицательное значение литерала, необходимо использовать унарный оператор “~“: ~Qx75, ~8. |

## 1.9 Объявление данных

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

Объявление без инициализации:

create integer x;

Объявление с инициализацией:

create string m='test';

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 KMO-2022 также допустима инициализация при объявлении. При этом переменной присваивается значение литерала или идентификатора, находящееся справа от знака равенства.

## 1.11 Инструкции языка

Все возможные инструкции языка программирования KMO-2022 представлены в общем виде в таблице 1.4.

Таблица 1.4 Характеристика инструкций языка KMO-2022

|  |  |
| --- | --- |
| Инструкция | Описание |
| Объявление переменной | create <тип данных><идентификатор>~  Пример: create integer w; |
| Возврат значения функции | return <идентификатор/литерал>~  Пример: return m; return 4; |
| Вывод данных | print <идентификатор/литерал/выражение>~  Пример: print 'hi'; print Func(4,3); print (2+2)\*2; |
| Вызов функции | <идентификатор>(<список параметров>)~  Пример: fi(z); fi(1); |
| Операция присваивания | <идентификатор>=<литерал/идентификатор/выражение>~  Пример: g=concatstr(a,b); |

## 1.12 Операции языка

Таблица 1.5 Характеристика операций языка KMO-2022

|  |  |
| --- | --- |
| Оператор | Приоритет |
| () | 0 |
| + (для данных типа integer) | 2 |
| - (бинарный) | 2 |
| ~ (унарный) | - |
| \* | 3 |
| / | 3 |
| % | 3 |

## 1.13 Выражения и их вычисления

Выражения языка KMO-2022 подчиняются следующим правилам:

1. Выражение, как и любая инструкция языка, должна заканчиваться сепаратором “;”;
2. Использование в одном выражении операндов разных типов недопустимо;
3. Разрешено изменять приоритет операций, используя оператор “()”;
4. Недопустимо использование двух арифметических операторов подряд;
5. Разрешено использовать в качестве операнда вызов функции;

## 1.14 Конструкции языка

Ключевые программные конструкции языка программирования KMO-2022 представлены в таблице 1.6

Таблица 1.6 Характеристика конструкций языка KMO-2022

|  |  |
| --- | --- |
| Конструкция | Характеристика |
| Главная функция | main  <  …  >; |

Продолжение таблицы 1.6

|  |  |
| --- | --- |
| Конструкция | Характеристика |
| Функция | <тип данных> function <идентификатор> (<тип данных> <идентификатор>, …)  <  …  return <идентификатор/литерал>;  >;  Функции пользователя должны быть определены вне тела главной функции. |
| Цикл | go <идентификатор> -> <литерал>;  <  …  >;  Инструкции в теле цикла повторяются пока идентификатор меньше указанного значения. |
| Строка комментария | $<комментарий> |

## 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

Семантические проверки конструкции языка программирования NRV-2022:

1. Проверка параметров функции на совпадение по типу и количеству.
2. Проверка совпадения типа функции и типа возвращаемого значения.
3. Проверка присваиваемых числовых значений на допустимость.
4. Проверка функций (проверка на рекурсию, объявление стандартных функций).
5. Проверка статических функций на параметры.
6. Проверка использования идентификатора до его объявления.
7. Проверка на тип присваиваемого значения.
8. Проверка на типы аргументов операций.
9. Проверка использования необъявленного идентификатора.
10. Проверка повторного объявления идентификатора.
11. Проверка на использование пустой строки.

## 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

В языке KMO-2022 присутствует стандартная библиотека, которая подключается автоматически при трансляции исходного кода в язык ассемблера. Все стандартные библиотеки реализованы на языке C++. Стандартные библиотеки подключены по умолчанию в программу. Вызовы стандартных функций доступны там же, где и вызов пользовательских функций. Также в стандартной библиотеке реализованы функции для манипулирования выводом, недоступные конечному пользователю. Содержимое библиотеки и описание функций представлено в таблице 1.7.

Таблица 1.7 Стандартная библиотека языка KMO-2022

|  |  |
| --- | --- |
| Функция | Характеристика |
| string concatstr(string a, string b) | Строковая функция, возвращающая результат конкатенации строк а и b, записанный в строку a |
| string 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;

В языке программирования KMO-2022 ввод данных не поддерживается.

Функции, управляющие вводом/выводом данных, реализованы на языке С++ и вызываются из транслированного кода, конечному пользователю недоступны. Пользовательская команда print в транслированном коде будут заменена вызовом нужных библиотечных функций. Библиотека, содержащая нужные процедуры, подключается на этапе генерации кода автоматически.

## 1.20 Точка входа

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

## 1.21 Препроцессор

В языке KMO-2022 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

В случае возникновения ошибки в исходном коде программы на языке программирования KMO-2022 и выявлении её транслятором в файл протокола выводится сообщение. Классификация обрабатываемых ошибок приведена в таблице 1.8.

Таблица 1.8 – Классификация сообщений транслятора

|  |  |
| --- | --- |
| Интервал | Описание ошибок |
| 0-99 | Системные ошибки |
| 100-112 | Ошибки при работе с файлами |
| 113-119 | Ошибки лексического анализа |
| 600-609 | Ошибки синтаксического анализа |
| 300-315 | Ошибки семантического анализа |

## 1.25 Контрольный пример

Контрольный пример представлен в приложении А.

# **2 Структура транслятора**

## 2.1 Компоненты транслятора, их назначение и принципы взаимодействия

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. Структура транслятора представлена на рисунке 2.1.

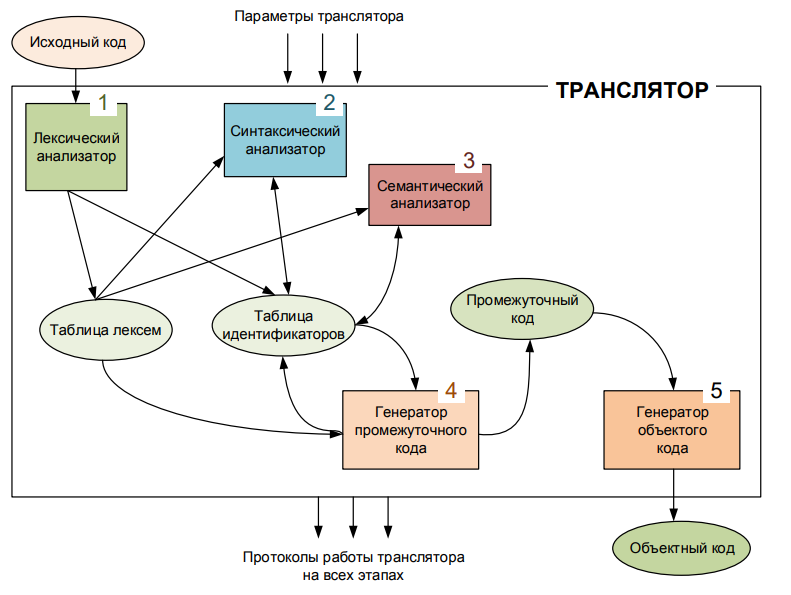


Рисунок 2.1 Структура транслятора

## 2.2 Перечень входных параметров транслятора

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

|  |  |  |
| --- | --- | --- |
| 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

В ходе работы программы формируются протоколы работы лексического, синтаксического и семантического анализаторов, которые содержат в себе перечень протоколов работы. В таблице 2.2 приведены протоколы, формируемые транслятором и их содержимое.

Таблица 2.2 Протоколы, формируемые транслятором языка KMO-2022

|  |  |
| --- | --- |
| 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. |

# **3 Разработка лексического анализатора**

## 3.1 Структура лексического анализатора

Входными данными является исходный код на языке KMO-2022. Таблица лексем и таблица идентификаторов, сформированные в результате лексического анализа, являются результатом работы лексического анализатора. Структура лексического анализатора представлена на рисунке 3.1.

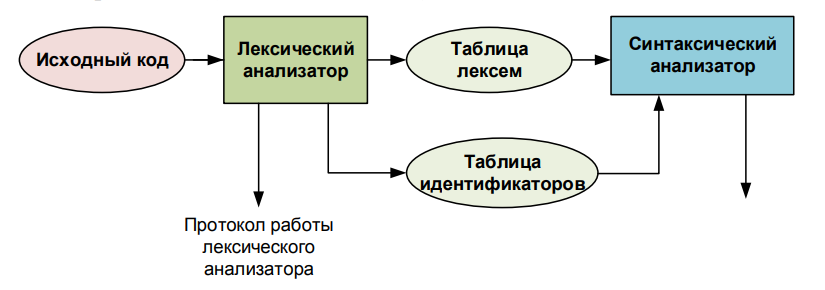


Рисунок 3.1 Структура лексического анализатора

## 3.2 Контроль входных символов

Принцип работы таблицы заключается в соответствии значения каждого её элемента значению в таблице Windows-1251.

Значения символов: F – запрещенные символы, Т – разрешенные символы, I – игнорируемые 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 I, то программа игнорирует его, если помечен как F – работа транслятора останавливается и выдаётся соответствующее сообщение об ошибке. Таблица входных символов представлена на рисунке 3.2.

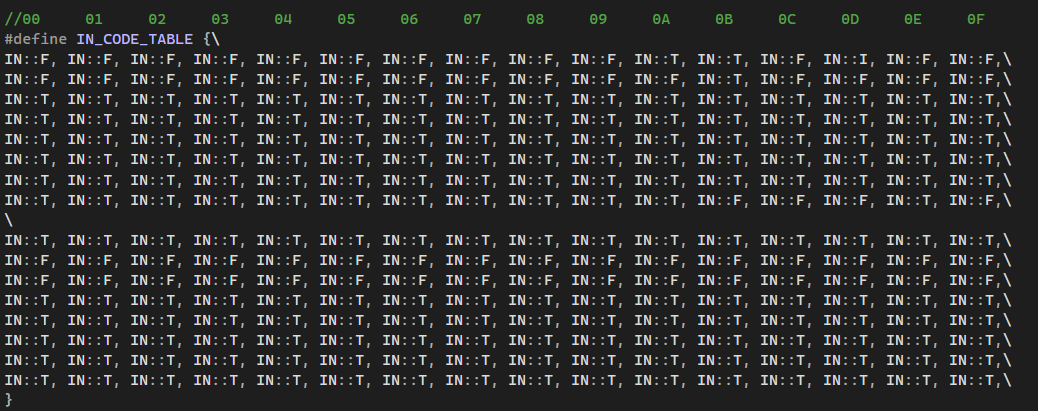


Рисунок 3.2 Таблица входных символов

## 3.3 Удаление избыточных символов

Избыточными символами являются пробелы.

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

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

Таблица 3.1 – Соответствие токенов и сепараторов с лексемами

|  |  |
| --- | --- |
| Токен | Лексема |
| integer | t |
| string | t |
| function | f |
| print | p |
| return | r |
| go | g |
| main | m |
| create | c |
| +-\*/% | v |
| = | = |
| -> | u |
| ( | ( |
| ) | ) |
| < | < |
| > | > |
| , | , |
| ; | ; |
| 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 |
| 113 | <Lexical>: Превышено максимальное количество строк в таблице лексем |
| 114 | <Lexical>: Превышено максимальное количество строк в таблице лексем |
| 115 | <Lexical>: Превышено максимальное количество строк в таблице идентификаторов |
| 116 | <Lexical>: Превышено максимальное количество строк в таблице идентификаторов |
| 117 | <Lexical>: Лексема не была распознана |
| 118 | <Lexical>: Превышена длина названия идентификатора |
| 119 | <Lexical>: Превышена длина лексемы |

# **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**

Входным параметром лексического анализатора является исходный текст программы, написанный на языке KMO-2022.

# **3.9 Алгоритм лексического анализа**

1. Считывание лексемы.
2. Распознавание лексемы с помощью конечных автоматов.
3. При удачном прохождении информация заносится в таблицу лексем и таблицу идентификаторов. Возврат к шагу 1.
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.

# **4 Разработка синтаксического анализатора**

## 4.1 Структура синтаксического анализатора

Синтаксический анализатор: часть компилятора, выполняющая синтаксический анализ, то есть исходный код проверяется на соответствие правилам грамматики. Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов. Выходной информацией– дерево разбора. Описание структуры синтаксического анализатора языка представлено на рисунке 4.1.

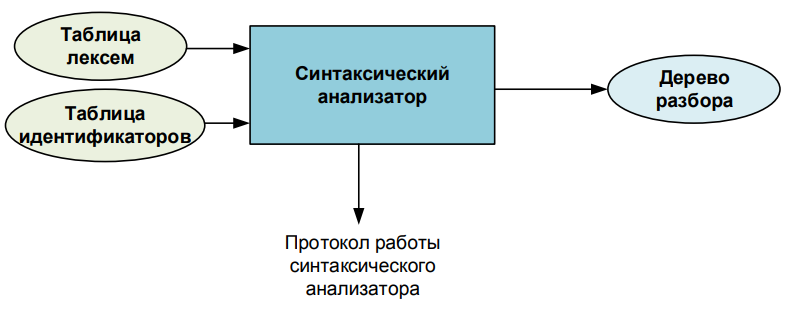


Рисунок 4.1 Структура синтаксического анализатора

## 4.2 Контекстно-свободная грамматика, описывающая синтаксис языка

В синтаксическом анализаторе транслятора языка KMO-2022 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 KMO-2022 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 KMO-2022. 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

Структура и перечень сообщений синтаксического анализатора представлены в таблице 4.3.

Таблица 4.3 – Структура и перечень сообщений синтаксического анализатора

|  |  |
| --- | --- |
| Код | Сообщение |
| 600 | <Syntactic>: Неверная структура программы. |
| 601 | <Syntactic>: Ошибочный оператор. |
| 602 | <Syntactic>: Ошибка в выражении. |
| 603 | <Syntactic>: Ошибка в параметрах функции. |
| 604 | <Syntactic>: Ошибка в параметрах вызываемой функции. |
| 605 | <Syntactic>: Ошибка в цикле. |
| 606 | <Syntactic>: Ошибка на правильность параметров вызываемой функции. |
| 607 | <Syntactic>: Ошибка на правильность тела цикла. |
| 608 | <Syntactic>: Ошибка в return. |
| 609 | <Syntactic>: Синтаксическая ошибка. Побробнее в log. |

## 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:

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

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

# **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

Функции, выполняющие проверку на этапе семантического анализа представлены в таблице 5.1.

Таблица 5.1 Функции, выполняющие проверку на этапе семантического анализа

|  |  |
| --- | --- |
| Функция | Характеристика |
| CheckParm | Проверка параметров функции на совпадение по типу и количеству |
| CheckReturn | 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 |
| CheckRecursion | Проверка функций на рекурсию |
| CheckPrint | Проверка операций в print |
| CheckStaticParm | Проверка функций на параметры статических функций |
| Semantic\_analysis | Семантический анализ |

## 5.3 Структура и перечень сообщений семантического анализатора

Сообщения, формируемые семантическим анализатором, представлены в таблице 5.2.

Таблица 5.2 Перечень сообщений семантического анализатора

|  |  |
| --- | --- |
| Код | Сообщение |
| 300 | <Semantic>: Количество параметров функции не совпадает. |
| 301 | <Semantic>: Типы формальных и фактических параметров функции не совпадают. |

Продолжение таблицы 5.2.

|  |  |
| --- | --- |
| Код | Сообщение |
| 302 | <Semantic>: Тип функции и тип возвращаемого значения не совпадают. |
| 303 | <Semantic>: Недопустимый тип присваиваемого значения. |
| 304 | <Semantic>: Операции над аргументами разных типов недопустимы. |
| 305 | <Semantic>: Число выходит за границы допустимого диапазона. |
| 306 | <Semantic>: Недопустимые операции со строками. |
| 307 | <Semantic>: Идентификатор не может быть использован до его объявления. |
| 308 | <Semantic>: Стандартные функции не требуют объявления. |
| 309 | <Semantic>: Использование необявленного идентификатора. |
| 310 | <Semantic>: Повторное обьявление уже обьявленного идентификатора. |
| 311 | <Semantic>: Нельзя вызывать функцию внутри этой же функции. |
| 312 | <Semantic>: Операции над типом string недопустимы. |
| 313 | <Semantic>: Ошибка в параметрах статической функции. |
| 314 | <Semantic>: Использование пустой строки запрещенно. |
| 315 | <Semantic>: Запрещенно в возврате функции вызывать функцию. |

## 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.

# **6 Преобразование выражений**

## 6.1. Выражения, допускаемые языком

В языке KMO-2022 допустимы выражения с использованием целочисленных литералов. 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 польской записи:

* исходная строка: выражение;
* результирующая строка: польская запись;
* стек: пустой;
* исходная строка просматривается слева направо;
* операнды переносятся в результирующую строку;
* операция записывается в стек, если стек пуст;
* операция выталкивает все операции с большим или равным приоритетом в результирующую строку;
* отрывающая скобка помещается в стек;
* закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются.

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

Пример преобразования выражения к польской записи представлен в таблице 6.4.

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% |

# **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. Структура генератора кода 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– .data и .const. Идентификаторы размещены в сегменте данных(.data). Литералы – в сегменте констант (.const). Соответствия между типами данных идентификаторов на языке KMO-2022 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка KMO-2022 и языка Ассемблера

|  |  |
| --- | --- |
| Тип идентификатора на языке KMO-2022 | Тип идентификатора на языке ассемблера |
| integer | SDWORD |
| string | Byte |

## 7.3 Статическая библиотека

В языке KMO-2022 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) | Строковая функция. Возвращает результат копирования строки str2 в str1, записанный в строку str1. |
| char\* concatstr(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 KMO-2022 . 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.

# **8 Тестирование транслятора**

## 8.1 Тестирование лексического анализатора

Результаты тестирования лексического анализатора показаны в таблице 8.1.

Таблица 8.1 – Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  <  #;  print 1;  >; | " <Lexical>: Лексема не была распознана" |

## 8.2 Тестирование синтаксического анализатора

Результаты тестирования синтаксического анализатора показаны в таблице 8.2.

Таблица 8.2 – Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  <  >; | “600: строка 1, <Syntactic>: Неверная структура программы.” |
| integer function f(integer )  <  create integer k=5;  return k;  >; | “ 603: строка 0, <Syntactic>: Ошибка в параметрах функции.” |
| main  <  create integer m=1\*/8;  print m;  >; | “602: строка 2, <Syntactic>: Ошибка в выражении.” |

## 8.3 Тестирование семантического анализатора

Результаты тестирования семантического анализатора показаны в таблице 8.3

Таблица 8.3 – Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |

Продолжение талбицы 8.3

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| create integer m;  create integer m; | "<Semantic>: Повторное обьявление уже обьявленного идентификатора." |
| string function concatstr(string str1, string str2)  <  print 'ok';  return 'ok';  >; | "<Semantic>: Стандартные функции не требуют объявления." |
| string function foo(string a)  <  create string t;  return 5;  >; | "<Semantic>: Тип функции и тип возвращаемого значения не совпадают." |

# **Заключение**

В ходе выполнения курсовой работы был разработан транслятор для языка программирования KMO-2022. Таким образом, были выполнены основные задачи данной курсовой работы:

1. сформулирована спецификация языка KMO-2022;
2. разработаны конечные автоматы и алгоритмы для реализация лексического анализатора;
3. разработана контекстно-свободная, приведённая к ослабленной нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
4. разработан семантический анализатор, осуществляющий проверку смысла используемых инструкций;
5. разработан транслятор с языка программирования KMO-2022 на язык Assembler;
6. проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка KMO-2022 включает:

1. 2 типа данных;
2. поддержку операции вывода;
3. возможность вызова функций стандартной библиотеки;
4. наличие 5 арифметических операторов для вычисления выражений;
5. оператор цикла;
6. структурированная систем для обработки ошибок пользователя.

# **Список использованных источников**

1. Курс лекций по КПО Наркевич А.С.

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

3. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. - 3-е изд. – Москва : Вильямс, 2003. - 429 с.

4. Прата, С. Язык программирования С++. Лекции и упражнения / С. Прата. – М., 2006 — 1104 c.

5. Страуструп, Б. Принципы и практика использования C++ / Б. Страуструп – 2009 – 1238 с

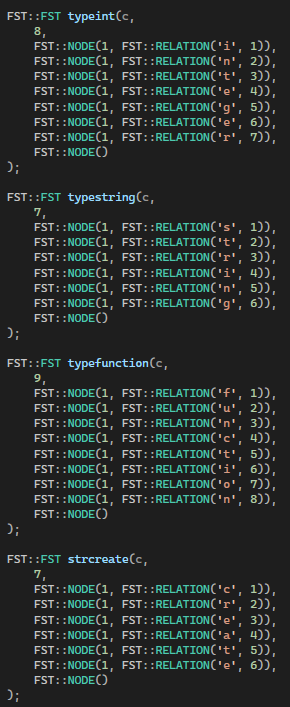
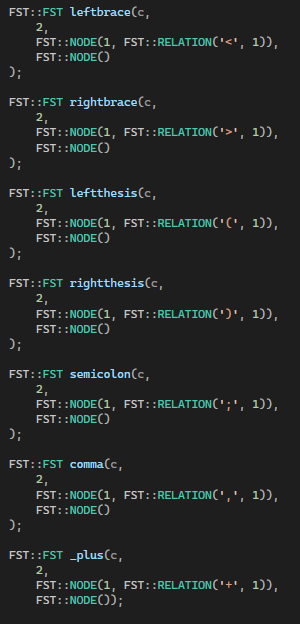
# **Приложение А**

Листинг 1 Исходный код программы на языке KMO-2022

|  |
| --- |
| $------------FUNCTIONS------------  integer function puj(integer ok)  <  create integer z= 5\* 6 - ok;  return z;  >;  string function fs ( string a, string b )  <  create string c3;  c3 = concatstr(a, b);  return c3;  >;  $------------MAIN------------  main  <  create integer i = ~25;  go i -> 10;  <  print i;  i = i +1;  >;  create string sa = '123456';  create string sb;  create string sc;  sb = '7890';  sc = fs(sa,sb);  print sc;  create integer tre;  tre = 27 / puj(4) + ( ~5);  print tre;  create string sbb = 'hello';  create string scc;  create string buf;  scc = 'tu rbo';  buf = copystr(sbb, scc);  print buf;  print ~Qx75;  print Bx011010;  >; |

**Приложение Б**

Фрагмент графов переходов конечных автоматов, соответствующие регулярным выражениям

# **Приложение В**

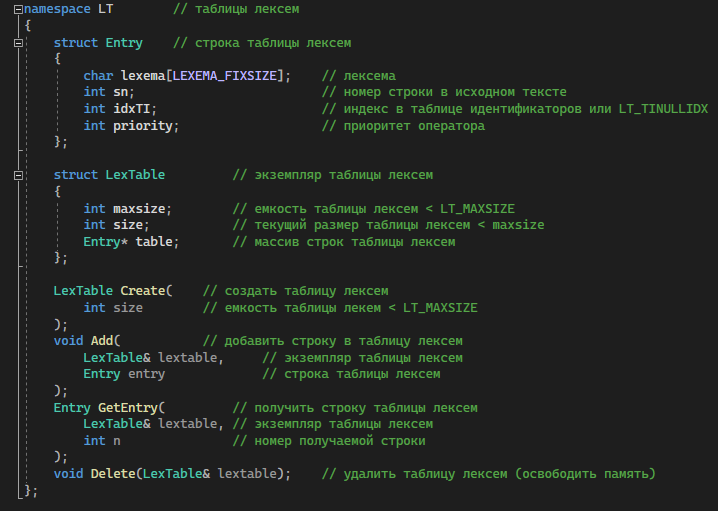


Рисунок 1. Структура таблицы лексем

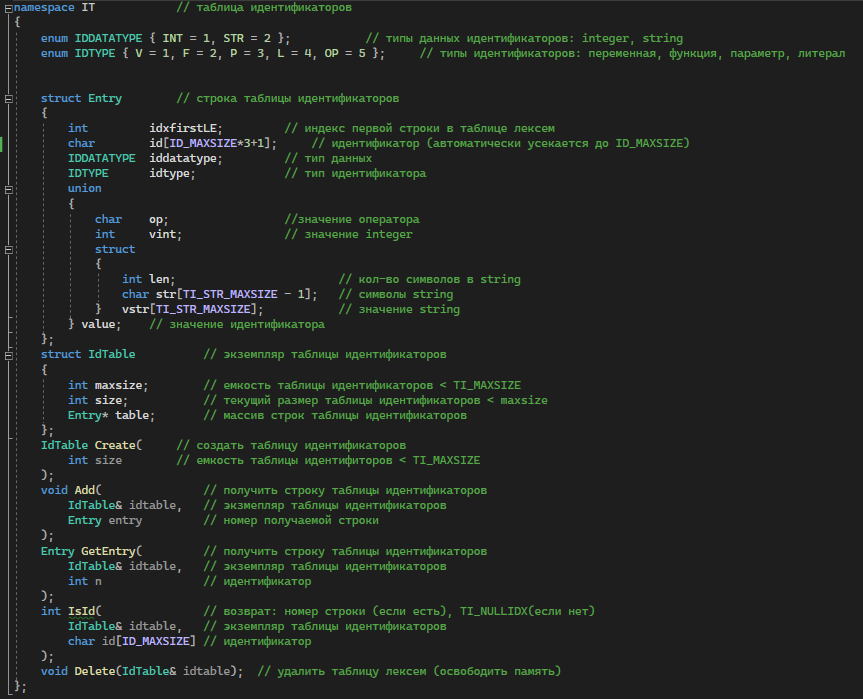


Рисунок 2. Структура таблицы идентификаторов

# **Приложение Г**

Таблица лексем:

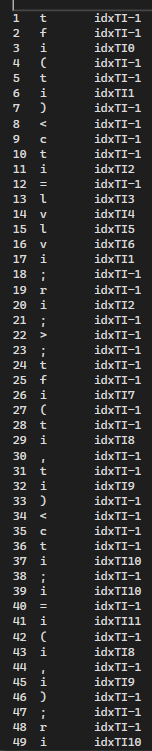
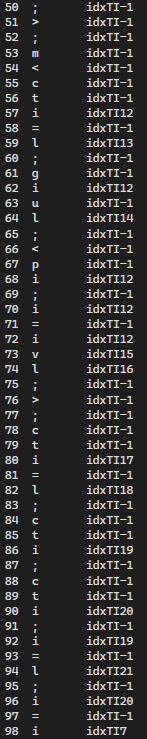
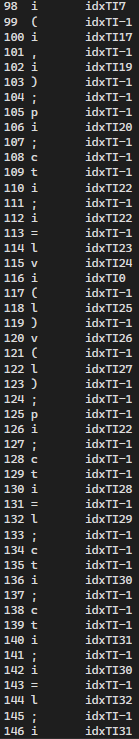
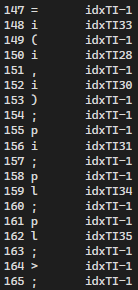
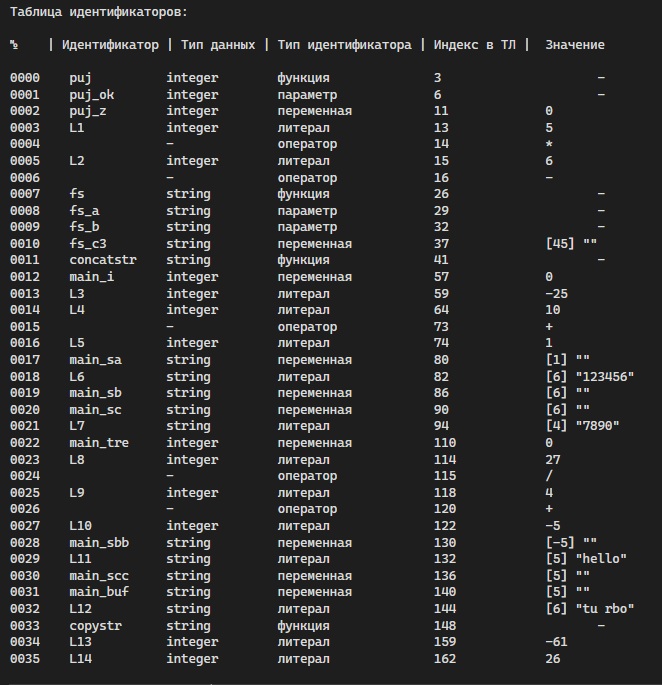
   

Таблица идентификаторов:



# **Приложение Д**

Таблица 4.1 Таблица правил переходов нетерминальных символов

|  |  |  |
| --- | --- | --- |
| Символ | Правила | Какие правила порождает |
| S | S->m<N>;  S->tfiP<NrX;>;S | Стартовые правила, описывающее общую структуру программы |
| N | N->cti;  N->cti;N  N->cti=E;  N->cti=E;N  N->pE;  N->pE;N  N->pE;  N->pE;N  N->i=E;  N->i=E;N  N->gZ;  N->gZ;N | Правила для конструкций в функциях |
| E | E->i  E->l  E->(E)  E->(E)M  E->i(W)  E->i()  E->i()M  E->iM  E->iM  E->i(W)M  E->lM | Правила для выражений |
| P | P->(F)  P->() | Правила для параметров функций |
| F | F->ti  F->ti,F | Правила для параметров при обьявлении функции |

Продолжение таблицы 4.1 Таблица правил переходов нетерминальных символов

|  |  |  |
| --- | --- | --- |
| Символ | Правила | Какие правила порождает |
| M | M->vE | Правила для арифметических действий |
| W | W->i  W->l  W->i,W  W->l,W | Правила для параметров вызываемой функции |
| Z | Z->iul;<J> | Правила для конструкции цикла |
| J | J->pE;  J->pE;J  J->i=E;  J->i=E;J | Правила для конструкций в теле цикла |
| X | X->l  X->i | Правила для выражений в return |

# **Приложение Е**

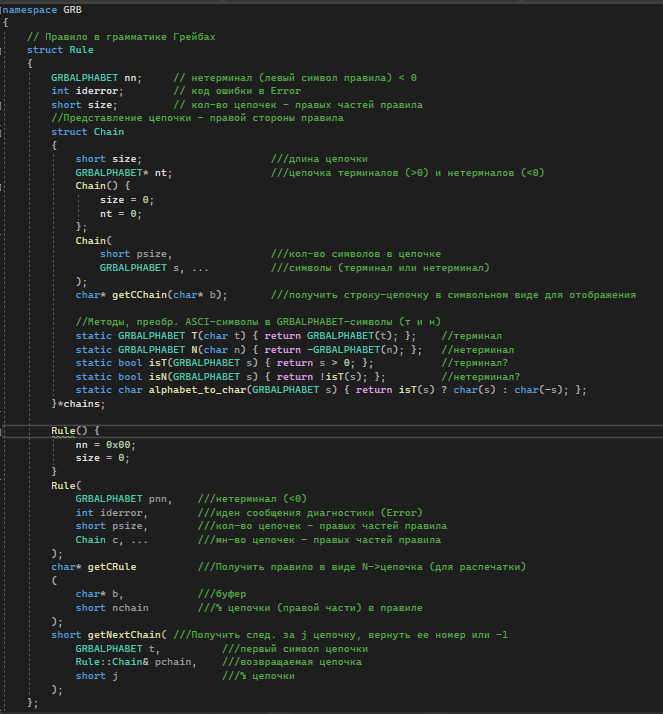


Рисунок 3. Структура правила в грамматике Грейбаха

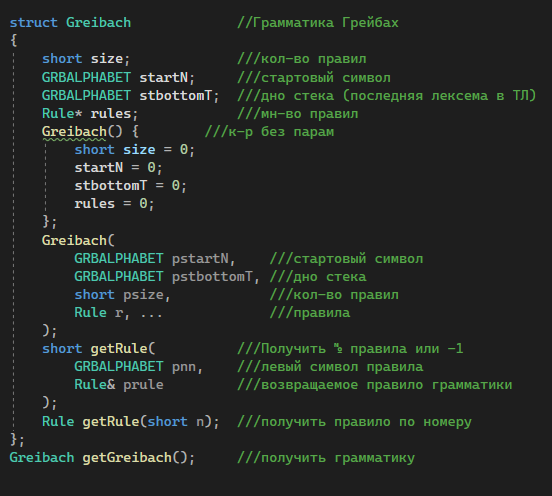
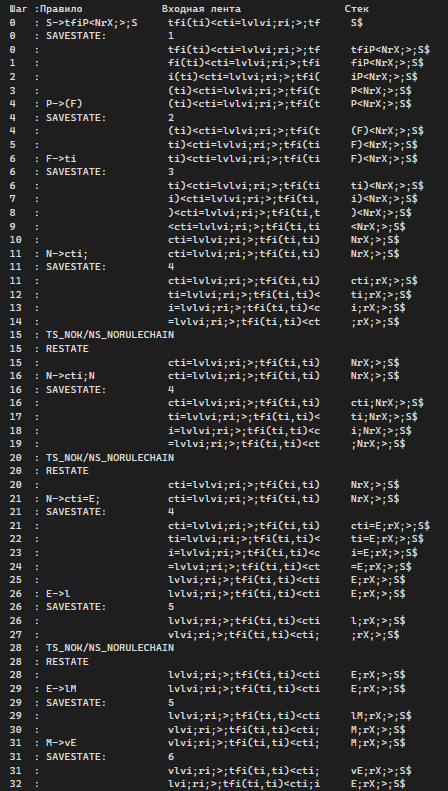


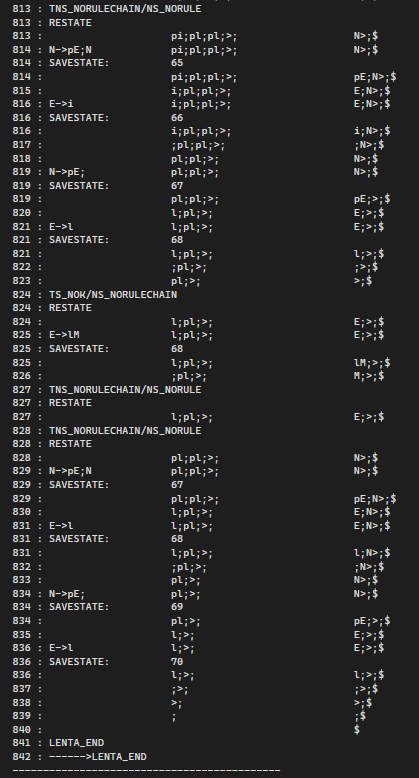
Рисунок 4. Структура грамматики Грейбаха

# **Приложение Ж**

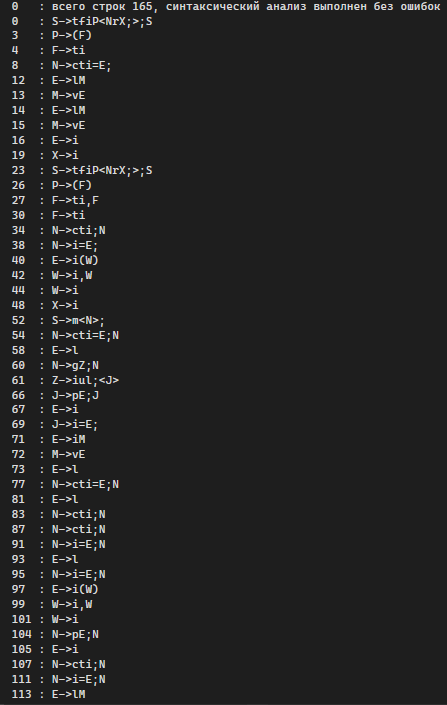
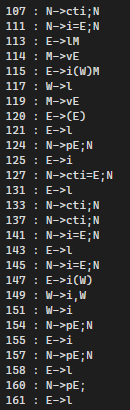
Начало разбора:



Конец разбора:



Дерево разбора:

# **Приложение З**

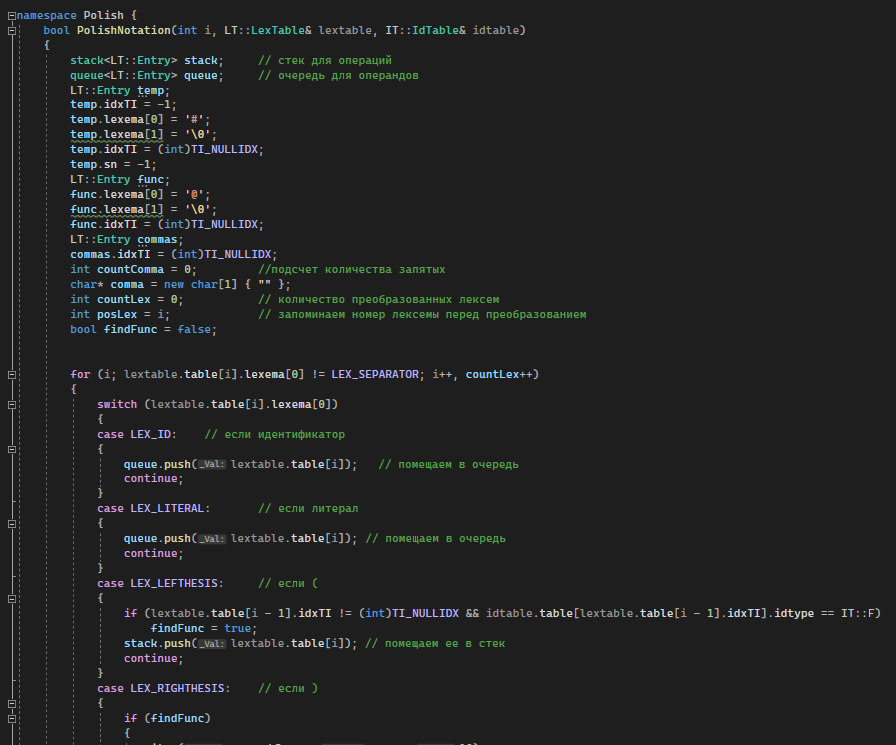


Рисунок 5. Программная реализация польской записи

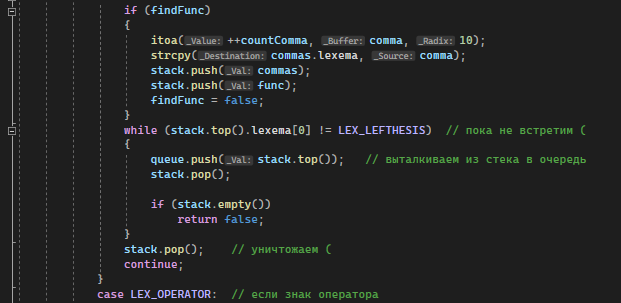


Рисунок 6. Программная реализация польской записи

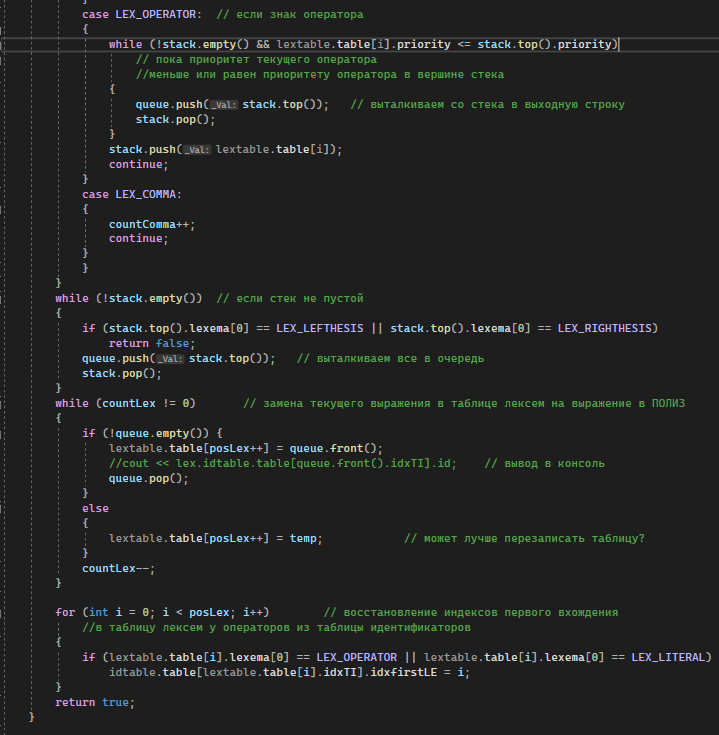
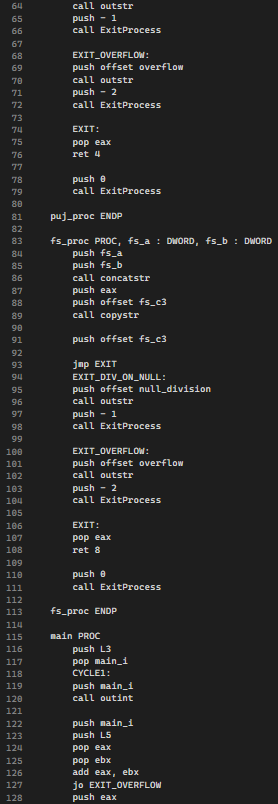
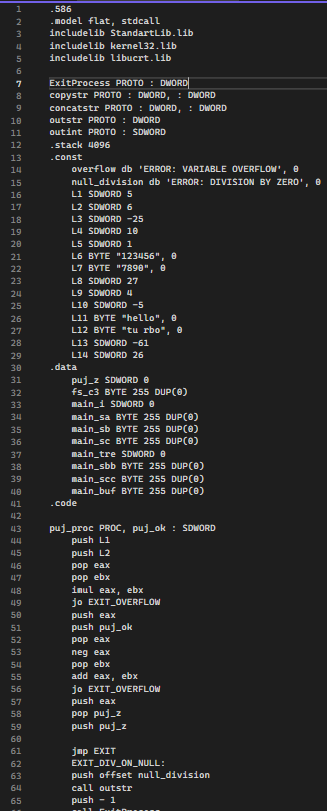


Рисунок 7. Программная реализация польской записи

# **Приложение И**

Р

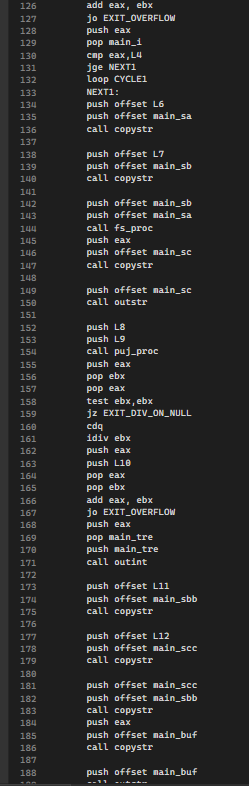
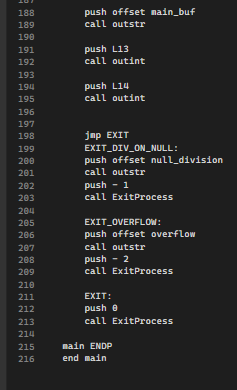
 

Рисунок 8. Результат генерации ассемблерного кода на основе контрольного примера