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 SII - 2020»

Выполнил студент Скородумов Иван Иванович

(Ф.И.О.)

Руководитель проекта пр.ст. Пахолко Алёна Степановна

(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 пр.ст. Пахолко Алёна Степановна

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

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

Нормоконтролер пр.ст. Пахолко Алёна Степановна

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

Курсовой проект защищен с оценкой

Минск 2020

**Содержание**

[**Глава 1. Спецификация языка программирования** 6](#_Toc501385916)

[**1.1. Характеристика языка программирования** 6](#_Toc501385917)

[**1.2.** **Алфавит языка** 6](#_Toc501385918)

[**1.3. Символы сепараторы** 7](#_Toc501385919)

[**1.4. Применяемые кодировки** 7](#_Toc501385920)

[**1.5. Типы данных** 7](#_Toc501385921)

[**1.6. Преобразование типов данных** 8](#_Toc501385922)

[**1.7. Идентификаторы** 8](#_Toc501385923)

[**1.8. Литералы** 8](#_Toc501385924)

[**1.9. Область видимости идентификаторов** 8](#_Toc501385925)

[**1.10. Инициализация данных** 9](#_Toc501385926)

[**1.11.** **Инструкции языка** 9](#_Toc501385927)

[**1.12. Операции языка** 10](#_Toc501385928)

[**1.13. Выражения и их вычисления** 10](#_Toc501385929)

[**1.14. Программные конструкции языка** 10](#_Toc501385930)

[**1.15.** **Область видимости** 10](#_Toc501385931)

[**1.16. Семантические проверки** 11](#_Toc501385932)

[**1.17.** **Распределение оперативной памяти на этапе выполнения** 11](#_Toc501385933)

[**1.18. Стандартная библиотека и её состав** 11](#_Toc501385934)

[**1.19.** **Ввод и вывод данных** 11](#_Toc501385935)

[**1.20. Точка входа** 12](#_Toc501385936)

[**1.21.** **Препроцессор** 12](#_Toc501385937)

[**1.22. Соглашения о вызовах** 12](#_Toc501385938)

[**1.23.** **Объектный код** 12](#_Toc501385939)

[**1.24. Классификация сообщений транслятора** 12](#_Toc501385940)

[**1.25.** **Контрольный пример** 12](#_Toc501385941)

[**Глава 2. Структура транслятора** 13](#_Toc501385942)

[**2.1 Компоненты транслятора, их назначение и принципы взаимодействия** 13](#_Toc501385943)

[**2.2 Перечень входных параметров транслятора** 14](#_Toc501385944)

[**2.3 Перечень протоколов, формируемых транслятором и их содержимое** 14](#_Toc501385945)

[**Глава 3. Разработка лексического анализатора** 15](#_Toc501385946)

[**3.1 Структура лексического анализатора** 15](#_Toc501385947)

[**3.2 Контроль входных символов** 15](#_Toc501385948)

[**3.3 Удаление избыточных символов** 16](#_Toc501385949)

[**3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов** 16](#_Toc501385950)

[**3.5 Основные структуры данных** 17](#_Toc501385951)

[**3.6 Принцип обработки ошибок** 17](#_Toc501385952)

[**3.7 Структура и перечень сообщений лексического анализатора** 18](#_Toc501385953)

[**3.8 Параметры лексического анализатора и режимы его работы** 18](#_Toc501385954)

[**3.9 Алгоритм лексического анализа** 18](#_Toc501385955)

[**3.10 Контрольный пример** 19](#_Toc501385956)

[**Глава 4. Разработка синтаксического анализатора** 19](#_Toc501385957)

[**4.1 Структура синтаксического анализатора** 19](#_Toc501385958)

[**4.2 Контекстно свободная грамматика, описывающая синтаксис языка** 20](#_Toc501385959)

[**4.3 Построение конечного магазинного автомата** 22](#_Toc501385960)

[**4.4 Основные структуры данных** 23](#_Toc501385961)

[**4.5 Описание алгоритма синтаксического разбора** 23](#_Toc501385962)

[**4.6 Структура и перечень сообщений синтаксического анализатора** 23](#_Toc501385963)

[**4.7 Параметры синтаксического анализатора и режимы его работы** 24](#_Toc501385964)

[**4.8 Принцип обработки ошибок** 24](#_Toc501385965)

[**4.9 Контрольный пример** 24](#_Toc501385966)

[**Глава 5. Разработка семантического анализатора** 25](#_Toc501385967)

[**5.1 Структура семантического анализатора** 25](#_Toc501385968)

[**5.2 Функции семантического анализатора** 25](#_Toc501385969)

[**5.3 Структура и перечень сообщений семантического анализатора** 25](#_Toc501385970)

[**5.4 Принцип обработки ошибок** 26](#_Toc501385971)

[**5.5 Контрольный пример** 26](#_Toc501385972)

[**Глава 6. Преобразование выражений** 26](#_Toc501385973)

[**6.1 Выражения, допускаемые языком** 27](#_Toc501385974)

[**6.2 Польская запись** 27](#_Toc501385975)

[**6.3 Программная реализация обработки выражений** 28](#_Toc501385976)

[**6.4 Контрольный пример** 28](#_Toc501385977)

[**Глава 7. Генерация кода** 29](#_Toc501385978)

[**7.1 Структура генератора кода** 29](#_Toc501385979)

[**7.2 Представление типов данных в оперативной памяти** 29](#_Toc501385980)

[**7.3 Алгоритм работы генератора кода** 29](#_Toc501385981)

[**Глава 8. Тестирование транслятора** 32](#_Toc501385982)

[**8.1 Тестирование фазы проверки на допустимость символов** 32](#_Toc501385983)

[**8.2** **Тестирование лексического анализатора** 32](#_Toc501385984)

[**8.3 Тестирование синтаксического анализатора** 33](#_Toc501385985)

[**8.4 Тестирование семантического анализатора** 33](#_Toc501385986)

[**Приложения** 34](#_Toc501385987)

[**Контрольный пример** 34](#_Toc501385988)

[**Приложение А** 35](#_Toc501385989)

[**Приложение В** 42](#_Toc501385990)

[**Приложение Г** 44](#_Toc501385991)

[**Приложение Е** 49](#_Toc501385992)

[**Литература** 51](#_Toc501385993)

# **Введение**

Целью курсового проекта поставлена задача разработки компилятора для моего языка программирования SII-2020. Этот язык программирования предназначен для выполнения простейших операций и арифметических действий над числами.

Компилятор SII-2020 – это программа, задачей которого является перевод программы, написанной на языке программирования SII-2020 в программу на язык ассемблера.

Транслятор SII-2020 состоит из следующих частей:

– семантический анализатор;

– синтаксический анализатор;

– логический анализатор;

– генератор исходного кода на языке ассемблера.

Исходя из цели курсового проекта, были определены следующие задачи:

– разбработка спецификации языка программирования;

– разбратка структуры транслятора;

– разработка лексического и семантического анализаторов;

– разработка синтаксического анализатора;

– преобразование выражений;

– генерация кода на язык ассемблера;

– тестирование транслятора.

Решения каждой из поставленных задач будут приведены в соответствующих главах курсового проекта.

# **Глава 1. Спецификация языка программирования**

## **Характеристика языка программирования**

Язык программирования SII-2020 классифицируется как процедурный, универсальный, строготипизированный, компилируемый и не объектно-ориентированный язык.

## **Алфавит языка**

Алфавит языка SII-2020 основан на кодировке Windows-1251, представленной на рисунке 1.1.



Рисунок 1.1 – Алфавит входных символов

На этапе выполнения могут использоваться символы латинского алфавита, цифры десятичной системы счисления от 0 до 9, спецсимволы, а также непечатные символы пробела, табуляции и перевода строки. Русские символы разрешены только в строковых литералах.

## **Символы сепараторы**

Символы, которые являются сепараторами представлены в таблице 1.1.

Таблица 1.1 – Сепараторы

|  |  |  |
| --- | --- | --- |
| Сепаратор | Название | Область применения |
| ‘ ‘ | Пробел | Допускается везде, кроме идентификаторов и ключевых слов |
| ; | Точка с запятой | Разделение конструкций |
| {…} | Фигурные скобки | Заключение программного блока |
| (…) | Круглые скобки | Приоритет операций, параметры функции |
| ‘…’ | Одинарные кавычки | Допускается везде, кроме идентификаторов и ключевых слов |
| = | Знак «равно» | Присваивание значения |
| , | Запятая | Разделение параметров |
| <<  >>  +  - | Знаки «сдвиг влево», «сдвиг вправо», «плюс», «минус» | Выражения |

## **Применяемые кодировки**

Для написания исходного кода на языке программирования SII-2020 используется кодировка Windows-1251.

## **Типы данных**

В языке SII-2020 реализованы два типа данных: целочисленный беззнаковый и символьный. Описание типов данных, предусмотренных в данным языке представлено в таблице 1.2.

Таблица 1.2 – Типы данных языка SII-2020

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| Целочисленный тип данных uint | Фундаментальный тип данных. Используется для работы с целочисленными положительными значениями. В памяти занимает 4 байта.  Максимальное значение: 65536.  Минимальное значение: 0.  Инициализация по умолчанию: значение 0. |
| Символьный тип данных symbol | Фундаментальный тип данных. Используется для работы с символами, каждый символ в памяти занимает 1 байт. Инициализация по умолчанию: символ конца строки “\0”. |

## **Преобразование типов данных**

Преобразование типов данных не поддерживается, т.е. язык является строготипизированным.

## **Идентификаторы**

В имени идентификатора допускаются только символы латинского алфавита нижнего регистра. Максимальная длина имени идентификатора - 6 символов. Максимальная длина имени идентификатора функции - 6 символов. При вводе идентификатора длиной более разрешенного количества символов, он будет усекаться. Имя идентификатора не может совпадать с ключевыми словами и не может иметь имя, как функция, уже содержащаяся в стандартной библиотеке.

## **Литералы**

С помощью литералов осуществляется инициализация переменных. В языке существует два типа литералов. Краткое описание литералов языка SII-2020 представлено в таблице 1.3.

Таблица 1.3 – Описание литералов

|  |  |  |  |
| --- | --- | --- | --- |
| Тип литерала | Регулярное выражение | Описание | Пример |
| Целочисленный литерал | [1-9]+[0-9]\* | Целочисленные неотрицательные литералы, по умолчанию инициализируются 0. Литералы могут быть только rvalue. | bind uint sum = 15;  15 – целочисленный литерал. |
| Символьный литерал | [a-z|A-Z|А-Я|а-я|0-9|!-/] | Символ, заключённый в ‘’ (одинарные кавычки), по умолчанию инициализируются пустой строкой. Литералы могут быть только rvalue. | bind symbol symbol = ‘T’;  T – символьный литерал. |

## **Область видимости идентификаторов**

Область видимости «сверху вниз» (по принципу С++). В языке SII-2020 требуется обязательное объявление переменной перед её инициализацией и последующим использованием. Все переменные должны находиться внутри программного блока. Имеется возможность объявления одинаковых переменных в разных блоках, т. к. переменные, объявленные в одной функции, недоступны в другой. Каждая переменная получает запись в специальное поле – название функции, в которой она объявлена.

## **Инициализация данных**

При объявлении переменной не допускается инициализация данных. Краткое описание способов инициализации переменных языка SII-2020 представлено в таблице 1.4.

Таблица 1.4 – Способы инициализации переменных

|  |  |  |
| --- | --- | --- |
| Конструкция | Описание | Пример |
| bind <тип данных> <идентификатор>; | Автоматическая инициализация: переменные типа uint инициализируются нулём, переменные типа symbol – пустым символом. | bind uint sum;  bind symbol chr; |
| <идентификатор> = <значение>; | Присваивание переменной значения. | sum = 15;  chr = ‘S’; |

## **Инструкции языка**

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

Таблица 1.5 – Инструкции языка программирования SII-2020

|  |  |
| --- | --- |
| Инструкция | Запись на языке SII-2020 |
| Объявление переменной | bind <тип данных> <идентификатор>; |
| Объявление функции | bind <тип данных> func <идентификатор> (<тип данных> <идентификатор>, …) {<блок кода>} |
| Присваивание | <идентификатор> = <значение>/<идентификатор>; |
| Объявление внешней функции | external <тип данных> <идентификатор> (<тип данных> <идентификатор>, …) |
| Блок инструкций | go  {  …  } |
| Возврат из подпрограммы | stop <идентификатор> / <литерал>; |
| Условная инструкция | until(условие) |
| Вывод целочисленных данных | sayi (<идентификатор> / <литерал>); |
| Вывод символьных данных | says (<идентификатор> / <литерал>); |

* 1. **Операции языка**

Язык программирования SII-2020 может выполнять операции, представленные в таблице 1.6.

Таблица 1.6 – Операции языка программирования SII‑2020

|  |  |  |  |
| --- | --- | --- | --- |
| Операция | Примечание | Типы данных | Пример |
| = | Присваивание | (uint, uint)  (char, char) | sum = 15;  chr = ‘T’; |
| < | Знак «меньше» для условной инструкции | (uint, uint) | if (sum < diff) {…}  else {…} |
| ( | Приоритет операций | - | sum = (a + b) \* c; |
| ) |
| + | Суммирование | (uint, uint) | sum = a + b; |
| - | Разность | (uint, uint) | diff = a – b; |
| << | Сдвиг влево | (uint, uint) | pr = a << b; |
| >> | Сдвиг вправо | (uint, uint) | pr = a >> b; |

## **Выражения и их вычисления**

Круглые скобки в выражении используются для изменения приоритета операций. Также не допускается запись двух подряд идущих арифметических операций. Выражение может содержать вызов функции, если эта функция уже содержится в стандартной библиотеке.

## **Программные конструкции языка**

Ключевые программные конструкции языка программирования SII-2020 представлены в таблице 1.7.

Таблица 1.7 – Программные конструкции языка SII-2020

|  |  |
| --- | --- |
| Конструкция | Запись на языке SII-2020 |
| Главная функция (точка входа) | go  {  …  stop <идентификатор> / <литерал>;  } |
| Функция | bondage <тип> <идентификатор> (<тип> <идентификатор>, …)  {  …  stop <идентификатор> / <литерал>;  } |

## **Область видимости**

В языке SII-2020 все переменные являются локальными. Они обязаны находится внутри программного блока функций (по принципу С++). Объявление глобальных переменных не предусмотрено. Объявление пользовательских областей видимости не предусмотрено.

## **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 возвращаемого значения должен совпадать с типом функции при её объявлении |
| 3 | Тип данных передаваемых значений в функцию должен совпадать с типом параметров при её объявлении |
| 4 | В функцию должны быть переданы параметры, если они ожидаются |
| 5 | Тип данных результата выражения должен совпадать с типом данных идентификатора, которому оно присваивается |

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

Все переменные размещаются в куче.

## **Стандартная библиотека и её состав**

Стандартная библиотека SII-2020 написана на языке программирования C++. Функции стандартной библиотеки с описанием представлены в таблице 1.9.

Таблица 1.9 – Состав стандартной библиотеки

|  |  |  |
| --- | --- | --- |
| Функция(C++) | Возвращаемое значение | Описание |
| int random(int) | uint | Возвращает случайное число |
| int ord(symbol) | symbol | Возвращает символьный код числа |

## **Ввод и вывод данных**

В языке SII-2020 не реализованы средства ввода данных.

Для вывода данных в стандартный поток вывода предусмотрены функции sayi (для данных целочисленного беззнакового типа) и says (для данных символьного типа), которые входят в состав стандартной библиотеки и описаны в таблице 1.9.

## **Точка входа**

В языке SII-2020 каждая программа должна содержать главную функцию suffer, т. е. точку входа, с которой начнется последовательное выполнение программы.

## **Препроцессор**

Препроцессор в языке программирования SII-2020 не предусмотрен.

## **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**

SII-2020 транслируется в язык ассемблера.

## **Классификация сообщений транслятора**

В случае возникновения ошибки в коде программы на языке SII-2020 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 |
| 0-99 | Системные ошибки |
| 100-109 | Ошибки параметров |
| 110-119 | Ошибки открытия и чтения файлов |
| 120-599 | Ошибки лексического анализа |
| 600-699 | Ошибки синтаксического анализа |
| 700-900 | Ошибки семантического анализа |

## **Контрольный пример**

Контрольный пример представлен в главе Приложения.

# **Глава 2. Структура транслятора**

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

Транслятор преобразует программу, написанную на языке SII-2020 в программу на языке ассемблера. Для указания выходных файлов используются входные параметры транслятора, которые описаны в пункте 2.2. 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 SII-2020 и формирование таблицы лексем и таблицы идентификаторов.

Семантический анализ в свою очередь является проверкой исходной программы SII-2020 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 и формирования промежуточного кода SII-2020. Для этого используются таблица лексем и идентификаторов. 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 SII-2020, прошедший все предыдущие этапы, 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 PAS-2017

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<имя\_файла> | Входной файл с расширением .txt, в котором содержится исходный код на PAS-2017. Данный параметр должен быть указан обязательно. В случае если он не будет задан, то выполнение этапа трансляции не начнётся. | Не предусмотрено |
| -log:<имя\_файла> | Файл содержит в себе краткую информацию об исходном коде на языке PAS-2017. В этот файл могут быть выведены таблицы идентификаторов, лексем, а также дерево разбора. | <имя\_файла>.log |
| -lex | Параметр для вывода таблицы лексем в файл “log.txt”. | Не выводится |
| -id | Параметр для вывода таблицы идентификаторов в файл “log.txt”. | Не выводится |
| -tree | Параметр для вывода дерева разбора в файл “log.txt”. | Не выводится |

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

Таблица с перечнем протоколов, формируемых транслятором языка PAS‑2017 и их назначением представлена в таблице 2.2.

Таблица 2.2 – Протоколы, формируемые транслятором языка PAS-2017

|  |  |
| --- | --- |
| Формируемый протокол | Описание протокола |
| Файл журнала, “log.txt” | Файл содержит в себе краткую информацию об исходном коде на языке PAS-2017. В этот файл могут быть выведены таблицы идентификаторов, лексем, а также дерево разбора. |
| “PASasm.asm” | Содержит сгенерированный код на языке Ассемблера. |

# **Глава 3. Разработка лексического анализатора**

## **3.1 Структура лексического анализатора**

Лексический анализатор – часть транслятора, выполняющая лексический анализ. Лексический анализатор принимает обработанный и разбитый на отдельные компоненты исходный код на языке PAS-2017. На выходе формируется таблица лексем и таблица идентификаторов. Структура лексического анализатора представлена на рисунке 3.1.



Рисунок 3.1 — Структура лексического анализатора PAS-2017

## **3.2 Контроль входных символов**

Исходный код на языке программирования PAS-2017 прежде чем транслироваться проверяется на допустимость символов. То есть изначально из входного файла считывается по одному символу и проверяется является ли он разрешённым.

Таблица для контроля входных символов представлена на рисунке 3.2.

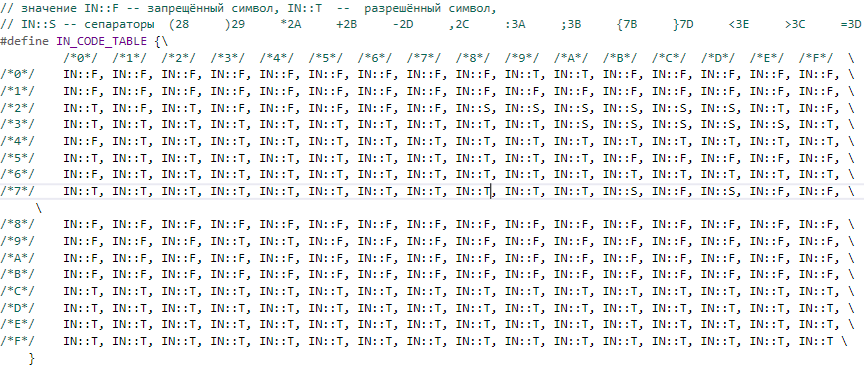


Рисунок 3.2 —Таблица контроля входных символов

Принцип работы таблицы заключается в соответствии значения каждому элементу в шестнадцатеричной системе счисления значению в таблице ASCII.

Описание значения символов: T – разрешённый символ, F – запрещённый символ, S – сепаратор.

## **3.3 Удаление избыточных символов**

Удаление избыточных символов не предусмотрено, так как после проверки на допустимость символов исходный код на языке программирования PAS-2017 разбивается на токены, которые записываются в очередь.

## **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 | Примечание | Цепочка | Лексема |
| Тип данных | Целочисленный тип данных | count | c |
| Строковый тип данных | road | s |
| Лексема | Объявление переменной | oops | v |
| Объявление функции | destiny | f |
| Возврат значения из функции | quintessence | r |
| Условная инструкция | if | u |
| else | e |
| Блок условной инструкции | [ | [ |
| ] | ] |
| Блок функции | { | { |
| } | } |
| Изменение приоритетности в выражении и отделение параметров функций | ( | ( |
| ) | ) |
| Сепараторы | ; | ; |

Продолжение таблицы 3.1

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | . | . |
| , | , |
| Оператор присваивания | = | = |
| Условный оператор | < | < |
| Оператор | Знаки арифметических операций | + | + |
| - | - |
| \* | \* |
| Идентификатор |  |  | i |
| Литерал |  | Целочисленный литерал | k |
|  | Строковый литерал | l |
| Точка входа |  | birth | h |
| Функция стандартной библиотеки | Возведение числа в степень | isqr | g |
| Степенной корень из числа | isqrt | a |
| Длина строки | strl | z |
| Вывод данных целочисленного типа | ququed | x |
| Вывод данных строкового типа | quques | y |

Каждому выражению соответствует детерминированный конечный автомат, то есть автомат с конечным состоянием, по которому происходит разбор данного выражения. На каждый автомат в массиве подаётся фраза и с помощью регулярного выражения, соответствующего данному графу переходов, происходит разбор. В случае успешного разбора выражения оно записывается в таблицу лексем. Если выражение является идентификатором или литералом, информация также заносится в таблицу идентификаторов. Пример реализации таблицы лексем представлен в приложении А.

Также в приложении А находятся конечные автоматы, соответствующие лексемам языка PAS-2017.

## **3.5 Основные структуры данных**

Основные структуры таблиц лексем и идентификаторов данных языка PAS-2017, используемых для хранения, представлены в приложении А. В таблице лексем содержится лексема, её номер, полученный при разборе, номер строки в исходном коде и индекс таблицы идентификаторов. В таблице идентификаторов содержится имя идентификатора, номер в таблице лексем, тип данных, тип идентификатора и его значение.

## **3.6 Принцип обработки ошибок**

При возникновении ошибки типа предупреждение транслятор продолжает свою работу, а предупреждения записываются в специальную структуру с номером ошибки и диагностическим сообщением.

Когда возникает критическая ошибка – работа транслятора прекращается.

В случаях, если на этапе было найдено менее 5 предупреждений, они будут выведены в файл после окончания этапа трансляции.

В случае, если была найдена критическая ошибка или в структуре находятся 5 предупреждений – работа транслятора прекращается и диагностическое сообщение будет выведено в файл.

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

Индексы ошибок, обнаруживаемых лексическим анализатором, находятся в диапазоне 120-125. Также сам текст ошибки содержит в себе префикс [LexA]. Перечень сообщений лексического анализатора представлен на рисунке 3.3.

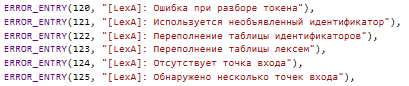


Рисунок 3.3 – Перечень ошибок лексического анализатора

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

Входные параметры используются для вывода результата работы лексического анализатора. Представлены в таблице 3.2.

Таблица 3.2 Входные параметры транслятора языка PAS-2017

|  |  |
| --- | --- |
| Входной параметр | Описание |
| -lex | Параметр для вывода таблицы лексем в файл протокола |
| -id | Параметр для вывода таблицы идентификаторов в файл протокола |

## **3.9 Алгоритм лексического анализа**

Алгоритм работы лексического анализа заключается в последовательном распознавании и разборе цепочек исходного кода и заполнение таблиц идентификаторов и лексем. Лексический анализатор производит распознаёт и разбирает цепочки исходного текста программы. Это основывается на работе конечных автоматов, которую можно представить в виде графов. В случае, если автомат не был подобран, запоминается номер строки, в которой находился этот токен и в последствии будет выведено сообщение об ошибке. Если токен разобран, то дальнейшие действия, которые будут с ним производиться, будут зависеть от того, чем он является. 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.

В случае, если токен является знаком арифметической операции либо функцией стандартной библиотеки, то он заносится в таблицу идентификаторов.

В случае, если является литералом, то заносится в таблицу идентификаторов с именем “literal<n>”, где n является номером литерала.

Когда встречаем токен, являющийся ключевым словом, которое отвечает за тип данных, заносим лексему, соответствующую ему, в таблицу лексем и запоминаем тип данных, которому он соответствует. В последствии, когда встречаем идентификатор, заносим его в таблицу идентификаторов с соответствующим ему типом данных и именем вида “aaaaabbbcc”, где a – символы из имени идентификатора, b – постфикс из имени функции, в которой он находится и c – номер функции.

Пример. Регулярное выражение для ключевого слова road: ‘road’.

Граф конечного автомата для этой лексемы представлен на рисунке 3.4. S0 – начальное состояние, S4 – конечное состояние автомата.

r

a

d

o

Рисунок 3.4 – Граф переходов для цепочки ‘road’

## **3.10 Контрольный пример**

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

# **Глава 4. Разработка синтаксического анализатора**

## **4.1 Структура синтаксического анализатора**

Синтаксический анализ – это фаза трансляции, выполняемая после лексического анализа и предназначенная для распознавания синтаксических конструкций PAS-2017. Входом для синтаксического анализа является таблица лексем и таблица идентификаторов, полученные после фазы лексического анализа. Выходом – дерево разбора. Структура синтаксического анализатора представлена на рисунке 4.1.



Рисунок 4.1 – Структура синтаксического анализатора PAS-2017

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

В синтаксическом анализаторе транслятора языка PAS-2017 используется контекстно-свободная грамматика , где

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

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

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

S – начальный символ грамматики, являющийся нетерминалом.

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

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

Грамматика языка PAS-2017 представлена в приложении Б.

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

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

Таблица 4.1 – Перечень правил, составляющих грамматику языка и описание нетерминальных символов PAS-2017

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| S | fci(F){N}S  fsi(F){N}S  h{N} | Порождает правила, описывающее общую структуру программы |
| N | vci;N  vsi;N  i=E;N  u(i<i)[N]N  u(i<k)[N]N  u(k<i)[N]N  u(k<k)[N]N  u(i<i)[N]Y  u(i<k)[N]Y  u(k<i)[N]Y  u(k<k)[N]Y | Порождает правила, описывающие конструкции языка |
| E | i  iM  l  lM  k  kM  g(k,i)  g(i,i)  g(i,k)  g(k,k)  a(k,i)  a(i,i)  a(i,k)  a(k,k)  z(i)  z(l)  g(k,i)M  g(i,i)M  g(i,k)M  g(k,k)M  a(i,i)M  a(i,k)M | Порождает правила, описывающие выражения |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
|  | i(W) |  |
| F | ci  ci,F  si  si,F | Порождает правила, описывающие параметры локальной функции при её объявлении |
| W | i  l  k  i,W  l,W  k,W | Порождает правила, описывающие принимаемые параметры функции |
| Y | e[N]N  e[N] | Порождает правила, описывающие обработку условной конструкции, если условие ложно |
| M | +E  -E  \*E  +(E)  -(E)  \*(E) | Порождает правила, описывающие знаки арифметических операций |

## **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. |

Продолжение таблицы 4.2

|  |  |  |
| --- | --- | --- |
|  | Начальное состояние автомата | Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики (нетерминальный символ А) |
|  | Начальное состояние магазина автомата | Символ маркера дна стека ($) |
|  | 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 конечного автомата и структуру грамматики Грейбах, описывающей правила языка PAS-2017. Данные структуры представлены в приложении В.

## **4.5 Описание алгоритма синтаксического разбора**

Принцип работы автомата следующий:

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

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

Перечень сообщений синтаксического анализатора представлен на рисунке 4.1.

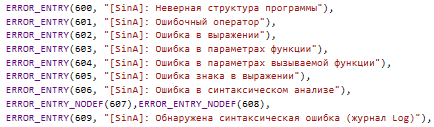


Рисунок 4.1 – Перечень сообщений синтаксического анализатора

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

Для управления результата работы синтаксического анализатора используются входные параметры, описанные в пункте 2.2Перечень входных параметров транслятора в таблице 2.1.

## **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 в протокол будет выведено диагностическое сообщение.

## **4.9 Контрольный пример**

Пример разбора синтаксическим анализатором исходного кода на языке PAS-2017 представлен в приложении Г. Дерево разбора исходного кода также представлено в приложении Г.

# **Глава 5. Разработка семантического анализатора**

## **5.1 Структура семантического анализатора**

Семантический анализ происходит при выполнении фазы лексического анализа и реализуется в виде отдельных проверок текущих ситуаций в конкретных случаях: установки флага или нахождении в особом месте программы (оператор выхода из функции, оператор ветвления, вызов функции стандартной библиотеки). Структура семантического анализатора представлена на рисунке 5.1.



Рисунок 5.1 — структура семантического анализатор

## **5.2 Функции семантического анализатора**

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

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

Все ошибка семантического анализатора имеют идентификатор свыше 700, а также префикс [SemA]. Сообщения, формируемые семантическим анализатором, представлены на рисунке 5.2.

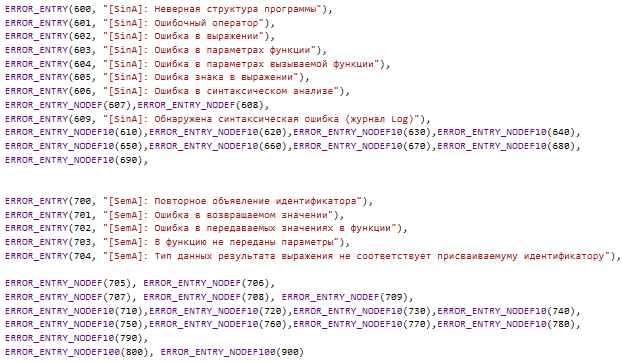


Рисунок 5.2 – Перечень сообщений семантического анализатора

## **5.4 Принцип обработки ошибок**

При обнаружении хотя бы одной ошибки транслятор проведёт только семантический анализ, после чего завершит свою работу. В случае обнаружения нескольких ошибок они будут записаны в массив. По окончании семантического анализа либо достижении максимального количества ошибок в массиве они будут выведены.

## **5.5 Контрольный пример**

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

**Глава 6. Преобразование выражений**

## **6.1 Выражения, допускаемые языком**

В языке PAS-2017 допускаются выражения, применимые к целочисленным типам данных. В выражениях поддерживаются арифметические операции, такие как +, -, \* и (), и вызовы функций как операнды арифметических выражений.

Приоритет операций представлен в таблице 6.1.

Таблица 6.1 – Приоритет операций в языке PAS-2017

|  |  |
| --- | --- |
| Операция | Значение приоритета |
| ( | 1 |
| ) | 1 |
| + | 2 |
| - | 2 |
| \* | 3 |

## **6.2 Польская запись**

Выражения в языке PAS-2017 преобразовываются к обратной польской записи.

Польская запись – это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок.

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

Алгоритм построения:

– читаем очередной символ;

– если он является идентификатором или литералом, то добавляем его к выходной строке;

– если символ является символом функции, то помещаем его в стек;

– если символ является открывающей скобкой, то она помещается в стек;

– исходная строка просматривается слева направо;

– если символ является закрывающей скобкой, то выталкиваем из стека в выходную строку все символы пока не встретим открывающую скобку. При этом обе скобки удаляются и не попадают в выходную строку;

– как только входная лента закончится все символы из стека выталкиваются в выходную строку;

– в случае если встречаются операции, то выталкиваем из стека в выходную строку все операции, которые имеют выше приоритетность чем последняя операция;

– также, если идентификатор является именем функции, то он заменяется на спецсимвол «@».

Таблица 6.2 – Пример преобразования выражения в обратную польскую запись

|  |  |  |
| --- | --- | --- |
| Исходная строка | Результирующая строка | Стек |
| (x+y)\*z |  |  |
| x+y)\*z |  | ( |
| +y)\*z | x |  |
| y)\*z | x | ( + |
| )\*z | x y | ( + |
| \*z | x y + |  |
| z | x y + | \* |
|  | x y + z | \* |
|  | x y + z \* |  |

## **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 необходимо для построения более простых алгоритмов их вычисления.

# **Глава 7. Генерация кода**

## **7.1 Структура генератора кода**

Генерация объектного кода — это перевод компилятором внутреннего представления исходной программы PAS-2017 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 PAS-2017 размещены в сегменте данных(.data). Литералы – в сегменте констант (.const). Соответствия между типами данных идентификаторов на языке PAS-2017 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка PAS-2017 и языка Ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке PAS-2017 | Тип идентификатора на языке ассемблера | Пояснение |
| count | SDWORD | Хранит целочисленный тип данных со знаком. |
| road | DWORD | Хранит указатель на начало строки. |
| literal(0-9) | BYTE  DWORD | Литералы: символьные,  целочисленные |

## **7.3 Алгоритм работы генератора кода**

Алгоритм генерации кода выглядит следующим образом:

1) Проходим полностью таблицу идентификаторов и заполняет поле .const литералами. Результат представлен на рисунке 7.2.

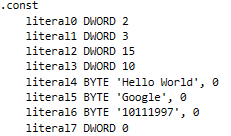


Рисунок 7.2 — Пример заполнения поля .const

2) Проходим таблицу лексем и ищем объявление переменных. В случае, если нашли литерал “v”, объявляем данную переменную в поле .data. Результат заполнения поля .data представлен на рисунке 7.3. После того, как данную переменную объявили, она удаляется из таблицы лексем, чтобы избежать переобъявления данного идентификатора в ассемблерном коде.

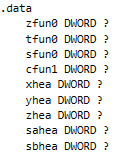


Рисунок 7.3 — Пример заполнения поля .data

3) Снова идём по таблице лексем и начинаем описывать функции и конструкции, написанные на языке PAS-2017. Конструкции if-else соответствует шаблон, по которому она и строится на языке ассемблера. В случае, если встречаем лексему “=”, описываем вычисление выражения. Описание алгоритма преобразования выражений представлено в пункте 7.3. Один из вариантов записи функции и конструкций внутри неё представлен на рисунке 7.4.

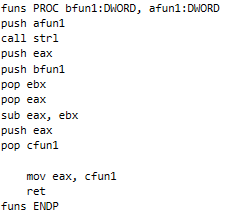


Рисунок 7.4 — Пример функции, полученной в результате генерации

## **Контрольный пример**

Генерируемый код записывается в файл «PASasm.asm». Сгенерированный код можно посмотреть в приложении Е.

# **Глава 8. Тестирование транслятора**

## **7г8.1 Тестирование фазы проверки на допустимость символов**

В языке PAS-2017 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 |
| birth  {ы} | 111: [IN]: Недопустимый символ в исходном файле (-in); |

## **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 |
| birth  {  oops count Q;  Q=2;  quintessence 0;  } | 120: [LexA]: Ошибка при разборе токена; Строка: 2  120: [LexA]: Ошибка при разборе токена; Строка: 3 |
| birth  {  x=5;  quintessence 0;  } | 121: [LexA]: Используется необъявленный идентификатор; Строка: 2 |
| destiny count fi(count y)  {  quintessence y;  } | 124: [LexA]: Отсутствует точка входа |
| birth  birth  {  quintessence 0;  } | 125: [LexA]: Обнаружено несколько точек входа |

## **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 |
| birth  {  oops count q;  q=2  quintessence 0;  } | Ошибка 609: [SinA]: Обнаружена синтаксическая ошибка (журнал Log) |

## **8.4 Тестирование семантического анализатора**

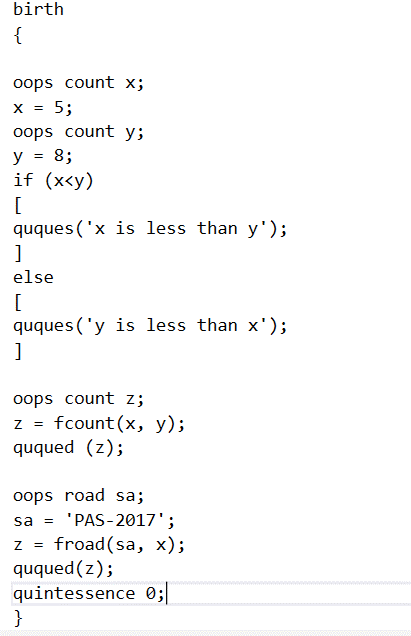
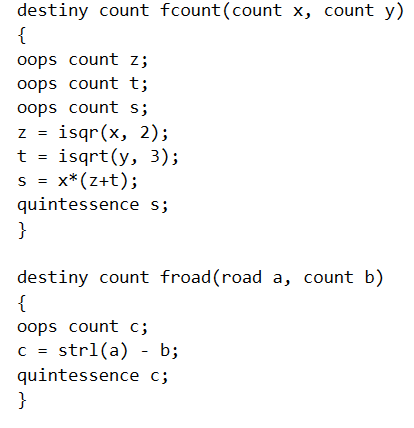
Итоги тестирования семантического анализатора приведены в таблице 8.4.

Таблица 8.4 – Тестирование семантического анализатора

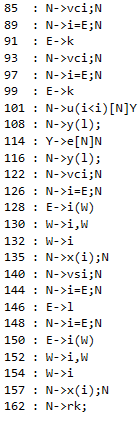
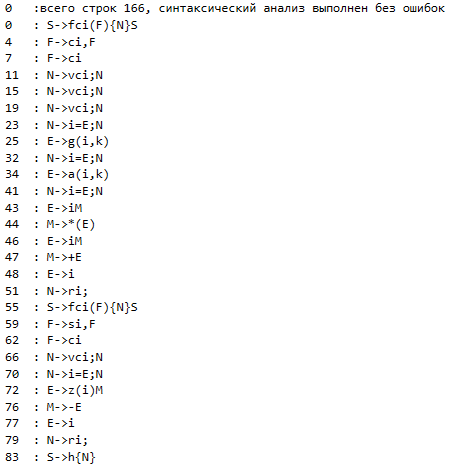
|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| birth  {  oops count q;  q=2;  oops road w;  w=q;  quintessence 0;  } | 704: [SemA]: Тип данных результата выражения не соответствует идентификатору; Строка 5 |
| birth  {  oops count q;  oops count q;  quintessence 0;  } | 700: [SemA]: Повторное объявление идентификатора; Строка: 13 |
| birth  {  oops count x;  x=5;  quques(x);  quintessence 0;  } | 702: [SemA]: Ошибка в передаваемых значениях в функции; Строка : 4 |

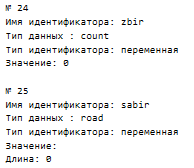
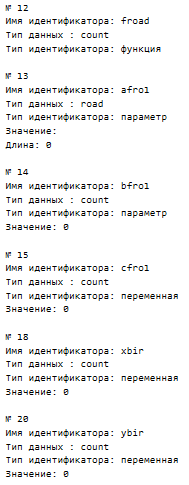
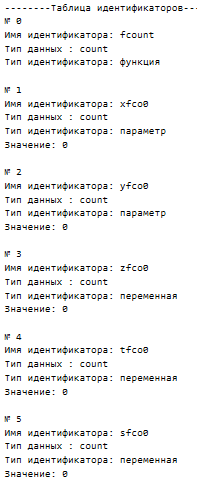
# **Приложения**

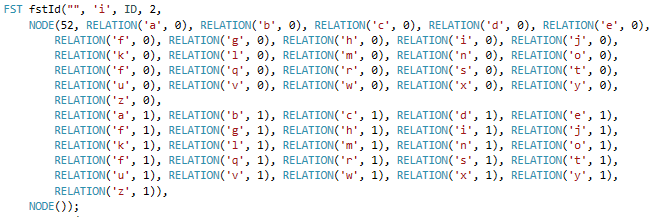
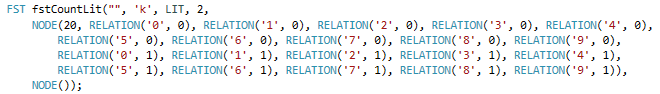
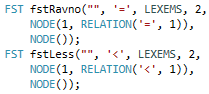
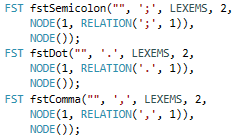
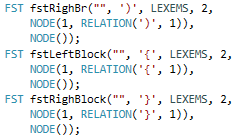
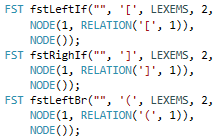
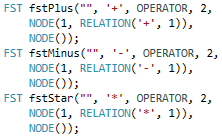
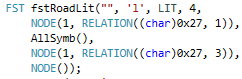
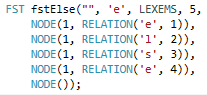
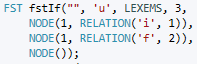
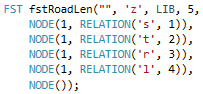
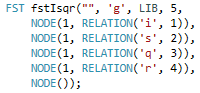
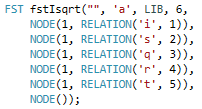
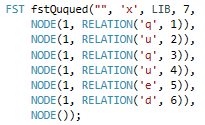
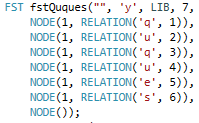
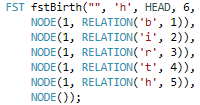
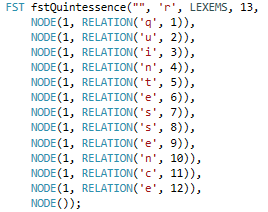
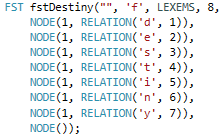
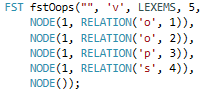
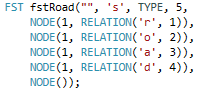
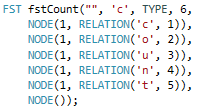
## **Контрольный пример**

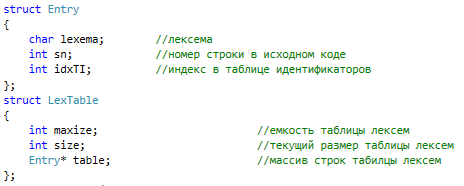


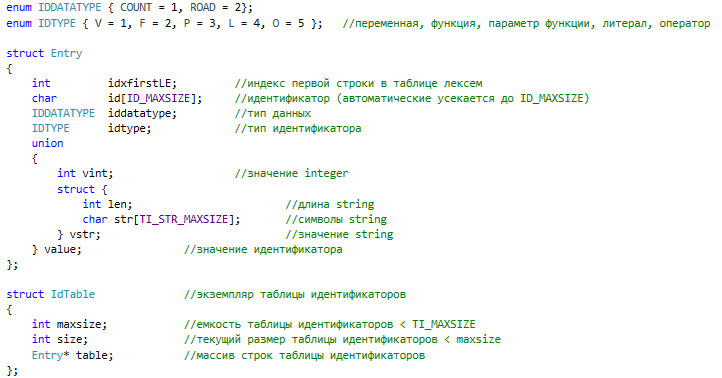
## **Приложение А**



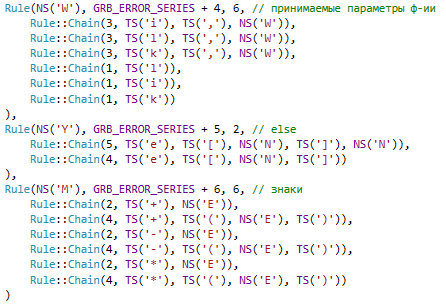
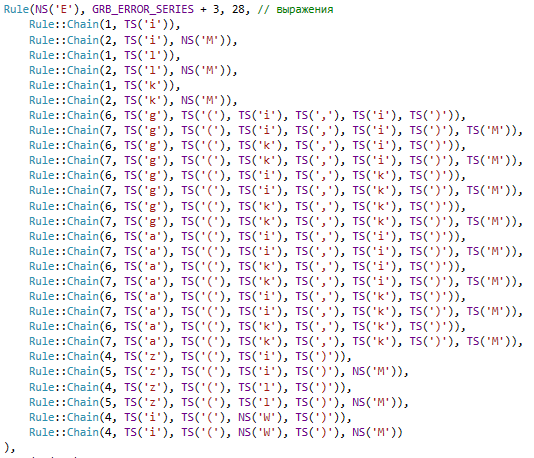
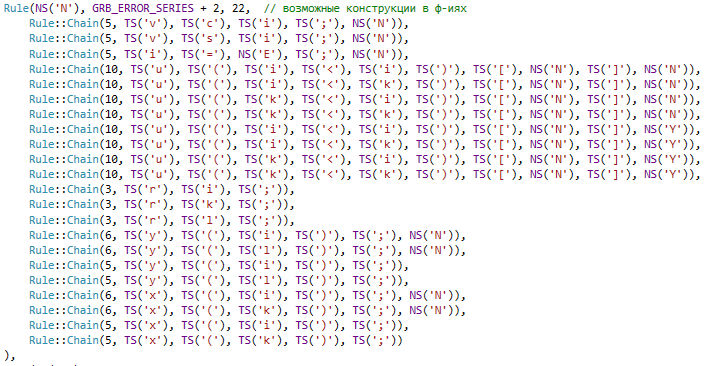
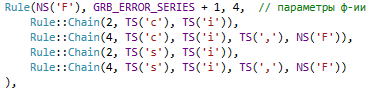
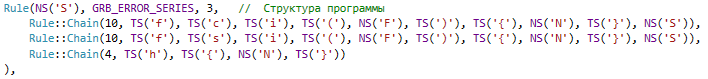






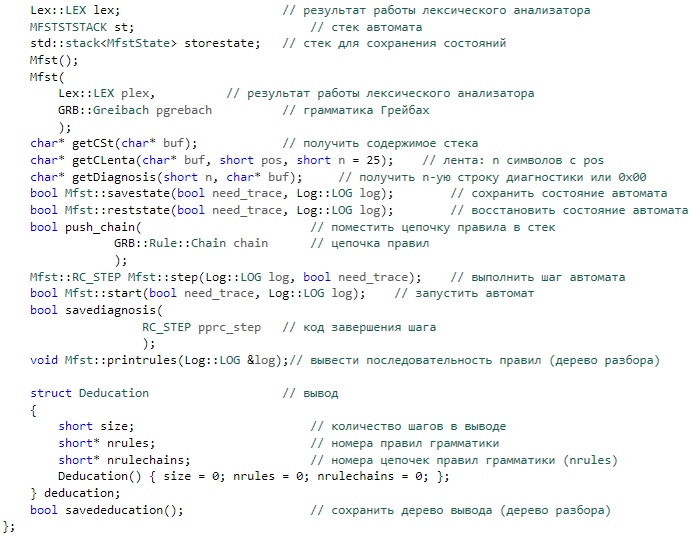


**Приложение Б**



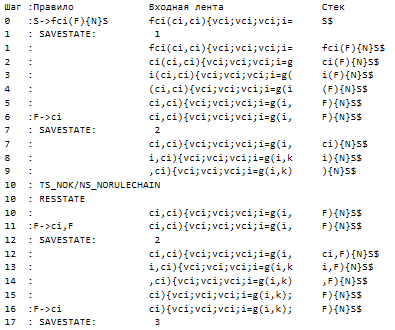
**Приложение В**



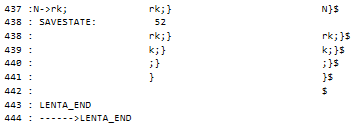


## **Приложение Г**

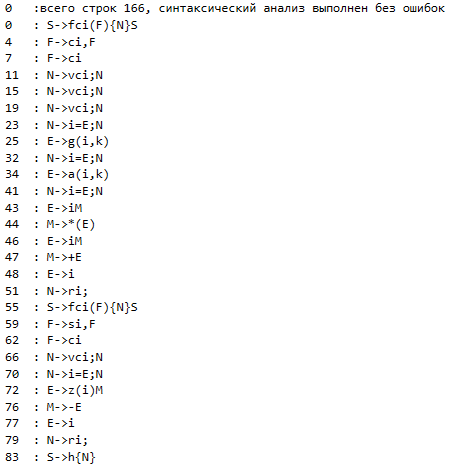
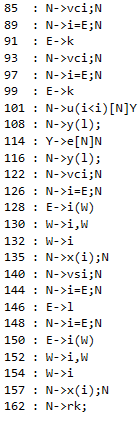
Начало разбора

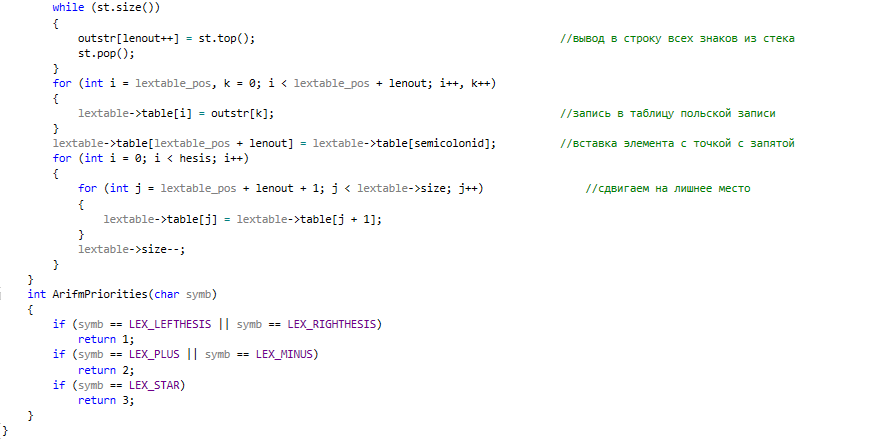
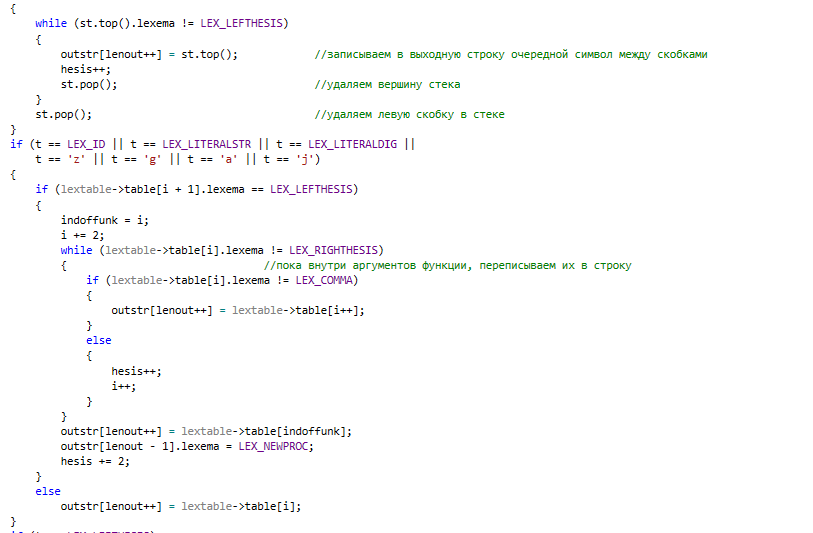
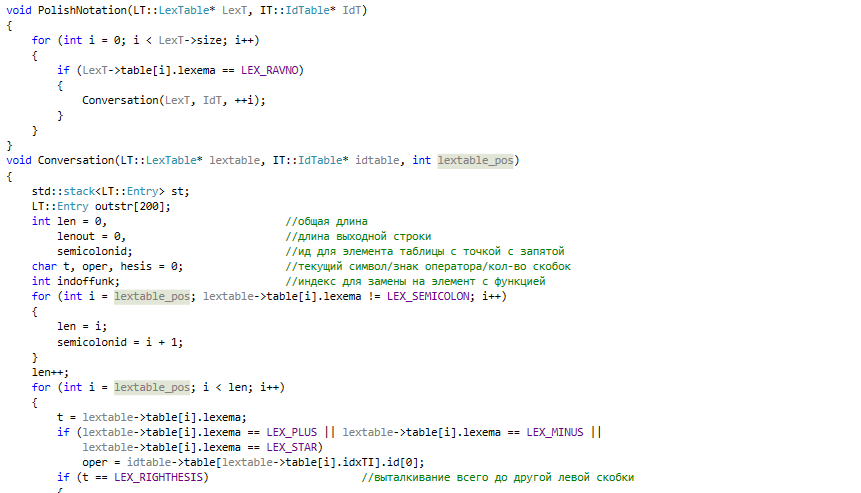


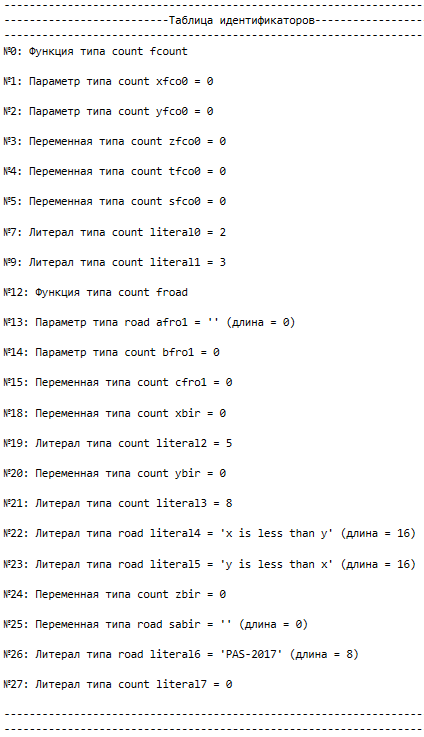
Конец разбора



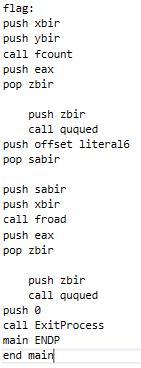
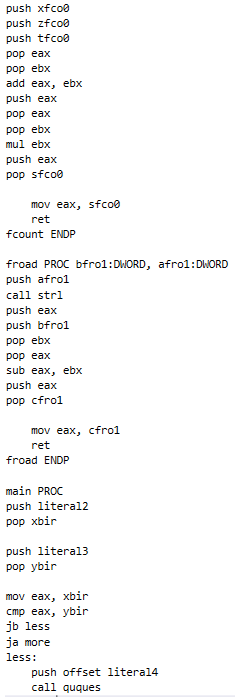
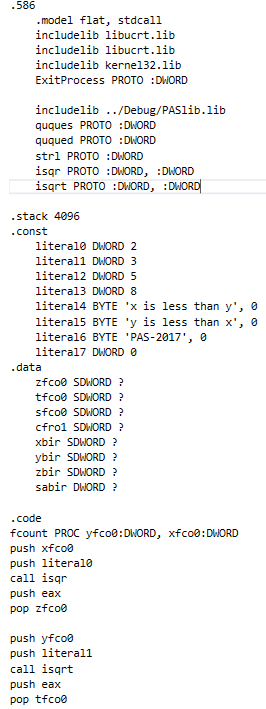
Правила разбора

**Приложение Д**





**Приложение Е**



# **Заключение**

По окончании выполнения всех пунктов, изложенных ранее, получили рабочий транслятор языка программирования PAS-2017 на язык ассемблера.

Язык PAS-2017 поддерживает 2 типа данных: целочисленный (count), строковый (road).

Для целочисленного типа реализована обработка 3 стандартных арифметических действий, скобок, обозначающих приоритет операций и условных скобок, 3 функций стандартной библиотеки .

Для строковых переменных реализованы 2 функции стандартной библиотеки.

На этапе семантического анализа производится проверка соответствия исходного кода спецификации по 5 правилам.

Реализованы 5 функции стандартной библиотеки.

Итоговое суммарное количество строк исходного кода ~2900.

# **Литература**

1. Ахо А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003. – 768с.
2. Ирвин К. Р. Язык ассемблера для процессоров Intel / К. Р. Ирвин. – M.: Вильямс, 2005. – 912с.