**Содержание**

[Введение 4](#_Toc58919967)

[1 Спецификация языка программирования 5](#_Toc58919968)

[1.1 Характеристика языка программирования 5](#_Toc58919969)

[1.2 Определение алфавита языка программирования 5](#_Toc58919970)

[2 Структура транслятора 15](#_Toc58919971)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 15](#_Toc58919972)

[2.2 Перечень входных параметров транслятора 16](#_Toc58919973)

[2.3 Перечень протоколов, формируемых транслятором и их содержимое 17](#_Toc58919974)

[3 Разработка лексического анализатора 18](#_Toc58919975)

[3.1 Структура лексического анализатора 18](#_Toc58919976)

[3.2 Контроль входных символов 18](#_Toc58919977)

[3.3 Удаление избыточных символов 19](#_Toc58919978)

[3.4 Перечень ключевых слов, сепараторов, символов операций, соответствующим им лексемам и конечных автоматов 19](#_Toc58919979)

[3.5 Основные структуры данных 20](#_Toc58919980)

[3.6 Принцип обработки ошибок 21](#_Toc58919981)

[3.7 Структура и перечень сообщений лексического анализатора 21](#_Toc58919982)

[3.8 Параметры лексического анализатора и режимы его работы 22](#_Toc58919983)

[3.9 Алгоритм лексического анализа 22](#_Toc58919984)

[3.10 Контрольный пример 23](#_Toc58919985)

[4 Разработка синтаксического анализатора 24](#_Toc58919986)

[4.1 Структура синтаксического анализатора 24](#_Toc58919987)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 24](#_Toc58919988)

[4.3 Построение конечного магазинного автомата 25](#_Toc58919989)

[4.4 Основные структуры данных 26](#_Toc58919990)

[4.5 Описание алгоритма синтаксического разбора 26](#_Toc58919991)

[4.6 Структура и перечень сообщений синтаксического анализатора 27](#_Toc58919992)

[4.7 Параметры синтаксического анализатора и режимы его работы 27](#_Toc58919993)

[4.8 Принцип обработки ошибок 27](#_Toc58919994)

[4.9 Контрольный пример 28](#_Toc58919995)

[5 Разработка семантического анализатора 30](#_Toc58919996)

[5.1 Структура семантического анализатора 30](#_Toc58919997)

[5.2 Функции семантического анализа 30](#_Toc58919998)

[5.3 Структура и перечень сообщений семантического анализатора 30](#_Toc58919999)

[5.4 Принцип обработки ошибок 31](#_Toc58920000)

[5.5 Контрольный пример 31](#_Toc58920001)

[6 Вычисление выражений 33](#_Toc58920002)

[6.1 Выражения, допускаемые языком 33](#_Toc58920003)

[7 Генерация кода 34](#_Toc58920004)

[7.1 Структура генератора кода 34](#_Toc58920005)

[7.2 Представление типов данных в оперативной памяти 34](#_Toc58920006)

[7.3 Библиотека 34](#_Toc58920007)

[7.4 Особенности алгоритма генерации кода 35](#_Toc58920008)

[7.5 Входные параметры генератора кода 37](#_Toc58920009)

[7.6 Контрольный пример 37](#_Toc58920010)

[8 Тестирование транслятора 38](#_Toc58920011)

[8.1 Общие положения 38](#_Toc58920012)

[8.2 Результаты тестирования 38](#_Toc58920013)

[Заключение 39](#_Toc58920014)

[Список использованных источников 40](#_Toc58920015)

[Приложение А 41](#_Toc58920016)

[Приложение Б 42](#_Toc58920017)

[Приложение В 45](#_Toc58920018)

[Приложение Г 46](#_Toc58920019)

[Приложение Д 48](#_Toc58920020)

[Приложение Е 51](#_Toc58920021)

[Приложение Ж 60](#_Toc58920022)

[Приложение И 61](#_Toc58920023)

# Введение

Целью курсового проекта является разработка собственного языка программирования ZEI-2020 и транслятора для него. Написание транслятора будет осуществляться на языке C++.

Транслятор — программа, которая преобразует исходный код на одном языке в исходный код на другом языке программирования. Его компонентами являются лексический, синтаксический и семантический анализаторы, а также генератор кода.

Исходя из цели, задачи на курсовой проект можно сформировать следующим образом:

* разработка языка программирования ZEI-2020, создание спецификации к нему (глава 1);
* разработка лексического анализатора (глава 3);
* разработка синтаксический анализатор (глава 4);
* разработка семантический анализатор (глава 5);
* разработка генератора кода, или интерпретатора в язык C# (глава 7);
* тестирование транслятора (глава 8).

# Спецификация языка программирования

1.1 Характеристика языка программирования

Язык ZEI-2020 — это универсальный язык высокого уровня. Он является строго типизированным, процедурным, интерпретируемым.

## **Определение алфавита языка программирования**

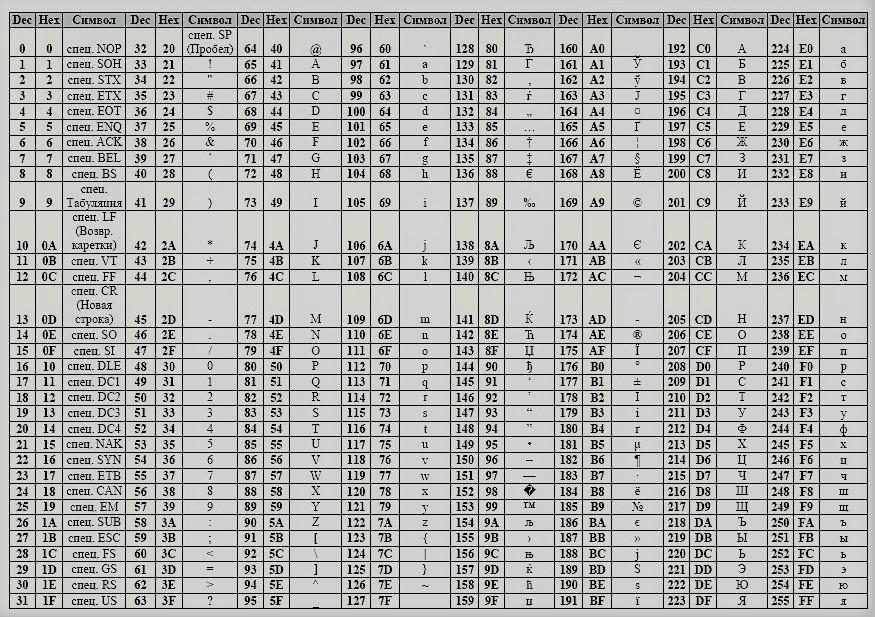
Базовый алфавит языка ZEI-2020 программирования использует символы восьмибитной кодировки ASCII, представленной на рисунке 1.1.

Рисунок 1.1 ASCII таблица кодов сиволов (Windows - 1251).

Для записи ключевых слов и конструкций языка используются строчные буквы латинского алфавита [a..z], а также символы-сепараторы, описанные в таблице 1.1.

Запрещённые символами языка являются все символы с десятичными кодами 0-8, 11-19, 127-255. При этом символы с кодами 224-255 (строчная кириллица) могут быть допущены в строковом литерале.

Допущенный в исходном коде запрещённый символ, при условии написания его вне строкового литерала, генерирует исключение, прекращая работу транслятора. Исключение также фиксируется в протоколе работы транслятора.

* 1. Применяемые сепараторы

Символы, входящие в число сепараторов, представлены в таблице 1.1.

Таблица 1.1 Сепараторы языка ZEI-2020.

|  |  |  |  |
| --- | --- | --- | --- |
| Код символа | Символ | Наименование | Описание |
| 32 |  | пробел | допускается везде, кроме  идентификаторов и ключевых слов; |
| 59 | ! | восклицательный знак | разделитель инструкций; |
| 40, 41 | (  ) | открывающая круглая скобка; закрывающая круглая скобка | используется для записи параметров функций (как фактические, так и формальные); условие в циклах или условных конструкциях; изменяют приоритетность операций; |
| 123, 125 | [  ] | открывающая квадратная скобка; закрывающая квадратная скобка | ограничение программного блока (тело функции, условной конструкции или цикла); |
| 44 | , | запятая | разделитель параметров функции; |
| 126 | ~ | тильда | однострочный комментарий (вся строка после знака игнорируется); |

* 1. Применяемые кодировки

Язык ZEI-2020 использует расширенную кодировку ACSII, описание которой представлено в подразделе 1.1.

* 1. Типы данных

В языке ZEI-2020 существуют типы данных, представленные в таблице 1.2.

Таблица 1.2 Типы данных языка ZEI-2020.

|  |  |
| --- | --- |
| Тип | Описание |
| tiny | целочисленный (1 байт), знаковый тип данных; диапазон значений: от -128 до 127; по умолчанию инициализируется нулем; |
| symbolic | строковый тип данных; может содержать строчные символы латиницы и кириллицы, а также пробелы, символы табуляции и символы знаков препинания (максимально 255 символов); по умолчанию инициализируется пустой строкой “”; |
| logical | логический тип данных, значения «true» или «false», по умолчанию инициализируется значением «false» |

Допустимые операции для каждого типа данных определены в пункте 1.12.

* 1. Преобразование типов данных

Язык ZEI-2020 не поддерживает преобразование типов данных.

* 1. Идентификаторы

Для именования функций, их параметров, а также переменных используются идентификаторы.

Идентификаторы в языке ZEI-2020 могут содержать лишь строчные символы латинского алфавита. Максимальное количество символов в идентификаторе — 15 (при превышении значения генерируется исключение). Идентификатор не должен совпадать с ключевыми словами языка.

<буква> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

<идентификатор> ::= <буква> { <буква> }.

Для стандартной библиотеки резервируются следующие идентификаторы:

* symbtotiny;
* symblen;
* sconcat.

Таким образом, примеры правильных идентификаторов языка ZEI-2020 — var, x, а неправильных — tiny (совпадает с ключевым словом), xxxxxxxxxxxxxxxx (превышено максимально допустимое количество символов).

1.8 Литералы

Литерал—элемент программы, который непосредственно представляет значение.

В языке ZEI-2020 предусмотрены типы литералов, описанные в таблице 1.3.

Для целочисленных литералов в разных представлениях предусмотрены постфиксы и префиксы, также указанные в таблице 1.3. до или после названия представления. Так, десятичное представление определяется как представление по умолчанию и постфикса не имеет, но оно может иметь префикс «m», определяющий, что число являеся отрицательным.

Таблица 1.3 Литералы языка ZEI-2020.

|  |  |  |
| --- | --- | --- |
| Тип | Представление, постфикс | Примечание |
| целочисленный | «m»десятичное,  двоичное «b»,  восьмеричное «q»,  шестнадцатеричное «h» | число в диапазоне от -128 до 127, интерпретируется как tiny; при выходе из диапазона генерируется исключение; |
| строковый | Последовательность или символов или символ; заключен в кавычки | строковый литерал, должен быть заключён в кавычки; интерпретируется как symbolic;  литералы «true» и «false» зарезервированы для типа данных logical |

Формально литералы языка ZEI-2020 можно определить следующим образом:

<буква> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | а | б | в | г | д | е | ж | з | и | ё | к | л | м | н | о | п | р | с | т | у | ф | х | ц | ч | ш | щ | ъ |ы |ь| э |ю | я

<знак>::= пробел | , | ? | : | ; | ! | - | табуляция

<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<строковый литерал>::= ”(<буква>|<цифра>|<знак>){ (<буква>|<цифра>|<знак>) }”

<целочисленный десятичный> := [m] {/<цифра>/}

<целочисленный двоичный>:= {/(0 | 1)/}b

<целочисленный восьмеричный>:= {/(0 | 1 | 2 |3 |4 |5 |6 |7)/}q

<целочисленный шестнадцатеричный>:={/(0|1 |2 |3 |4 |5 |6 |7 |8|9|A|B|C|D|E|F )/}h

Примеры правильных литералов языка ZEI-2020 — 10, 39h, 15q, 11b, “abc”, а неправильных — 9123, 0x14, m12q, “ABC”.

* 1. Объявление данных

Объявление идентификаторов в языке происходит с помощью ключевого слова set. Обязательно указание типа данных. Не обязательно присваивать значение при объявлении. Запрещено использование необъявленных идентификаторов или переобъявление. Возможно объявление одинаковых идентификаторов в разных областях видимости. Объявление глобальных переменных не предусмотрено. Объявление идентификаторов внутри условных конструкций, вне блока функции, в теле условных конструкций, а также в теле цикла не допускается.

Формально объявление идентификатора можно описать следующим образом:

set<тип данных> <идентификатор>[:(<литерал>|<идентификатор>| <выражение>)]!

1.10 Инициализация данных

Присвоение значение идентификатора допускается при его объявлении. При этом переменной будет присвоено значение литерала или идентификатора, стоящего справа от знака присваивания. Объектами-инициализаторами могут быть идентификаторы, литералы, выражения и вызовы функций. Предусмотрены значения по умолчанию, если переменные не инициализированы: значение 0 для целочисленного типа данных и строка нулевой длины ("") для строкового.

* 1. Инструкции языка

Язык ZEI-2020 предусматривает инструкции, описанные в таблице 1.4. Инструкции языка разделяются символом «!».

Таблица 1.4 Инструкции языка ZEI-2020.

|  |  |
| --- | --- |
| Инструкция | Описание |
| set<типданных><идентификатор>:<значение>! | объявление переменной |
| <тип данных> func <идентификатор>(<формальные параметры>)[ ] | объявление функций |
| show(<идентификатор>|<литерал>)! | вывод данных |
| <идентификатор>:<литерал>!  <идентификатор>:<выражение>!  <идентификатор1>:<идентификатор2>! | присвоить значение |
| <идентификатор функции>(<фактические параметры>)! | вызов функции |
| giveback <идентификатор>|<литерал>! | возврат из функции |

* 1. Операции языка

В языке ZEI-2020 предусмотрены операции, описанные в таблице 1.5.

Среди арифметических операций наибольшую приоритетность операций операции сложения и деления. При одинаковом приоритете первой выполнится операция, расположенная левее. Изменить приоритетность можно с помощью круглых скобок.

Таблица 1.5 Операции языка ZEI-2020.

|  |  |  |
| --- | --- | --- |
| Группа | Операторы | Типы данных |
| Логические | 1. > (бинарный) – оператор «больше»;  2.< (бинарный) – оператор «меньше»;  3.=(бинарный) – оператор проверки на равенство;  4.^(бинарный) – оператор проверки на неравенство; | tiny;  = и ^ можно применять и к типу logical; |
| Бинарные (сдвиги) | 1. \ – сдвиг влево;  <идентификатор|литерал>:<идентификатор|литерал>\<литерал>!   1. / – сдвиг вправо;   <идентификатор|литерал>:<идентификатор|литерал>/<литерал>! | tiny; |
| Арифметические | 1. + (бинарный) – оператор сложения; 2. - (бинарный) – оператор вычитания; 3. \*(бинарный) – оператор произведения; 4. # (бинарный) – оператор деления; | tiny. |

* 1. Выражения и их вычисление

В выражениях языком ZEI-2020 предусмотрено использование операций, описанных в таблице 1.5.

Предусмотрены следующие правила составления выражений:

* + выражения читаются слева направо и записываются в одну строку;
  + для изменения приоритета операция используются круглые скобки;
  + допустимо использование арифметических выражений с использованием вызова функции.
  1. Конструкции языка

В языке ZEI-2020 можно использовать конструкции языка, описанные в таблице 1.6.

Таблица 1.6 Конструкции языка ZEI-2020.

|  |  |
| --- | --- |
| Конструкция | Реализация, описание |
| 1 | 2 |
| Главная функция, точка входа в программу | perform  [  инструкции  ] |
| Функция | <тип данных> func <идентификатор>(<формальные параметры>)  [  инструкции  giveback <идентификатор/литерал>!  ] |
| Оператор цикла | loop**(**условие **)**  **[**  инструкции  **]**  Цикл (операторы внутри блока loop) выполняется, пока истинно условие. |
| 1 | 2 |
| Условная конструкция | when(условие )  [  инструкции1  ]  otherwise  [  инструкции2  ]  Если условие истинно, выполняются инструкции, определенные в блоке when, если ложно — определенные в блоке otherwise. Также блок otherwise может быть опущен. |

* 1. Область видимости идентификаторов

В языке ZEI-2020 область видимости сверху вниз (по принципу С++). Передаваемые в функцию параметры видны только внутри функции; идентификаторов, объявленные внутри функций, доступны лишь внутри той же функции. Запрещено объявление одинаковых идентификаторов в одной области видимости.

Идентификаторы как библиотечных, так и определенных пользователем функций имеют глобальную видимость (GLB). Наличие переменных с глобальной областью видимости не предусмотрено.

* 1. Семантические проверки

В языке ZEI-2020 предусмотрены следующие семантические проверки:

* + наличие функции perform– точки входа в программу;
  + единственность точки входа в программу perform;
  + переопределение идентификаторов;
  + использование идентификаторов без их объявления;
  + объявление функций вне других функций (в том числе perform);
  + проверка соответствия типа функции и возвращаемого значения;
  + ограничение количества максимально возможных параметров определяемой функции;
  + корректность передаваемых в функцию параметров: количество, типы;
  + выход за пределы диапазона числовых литералов или превышение размера строковых литералов;
  + корректность составленного условия цикла или условного оператора;
  + корректность выражений и арифметических операций (типы данных, допустимые операторы).
  1. Стандартная библиотека и её состав

В языке ZEI-2020 существует возможность использовать стандартную библиотеку, которая автоматически подключается на этапе генерации кода. Её функции описаны в таблице 1.7.

Вызов стандартных функций доступен там же, где и вызов пользовательских функций.

Таблица 1.7 Стандартная библиотека и её состав.

|  |  |  |
| --- | --- | --- |
| Функция стандартной библиотеки | Описание | Тип возвращаемого значения |
| symbtotiny (<symbolic идентификатор / литерал >) | преобразование строки в число | tiny |
| symblen (<symbolic идентификатор>) | вычисление длины строки | tiny |
| gettime (<symbolic идентификатор / литерал >) | добавление к строке текущего времени | symbolic |
| genertiny() | получение случайного числа из диапазона [-128; 127] | tiny |
| generlogical() | получение случайного значения true или false | logical |

* 1. Ввод и вывод данных

В языке ZEI-2020 поток ввода не предусмотрен.

Вывод данных осуществляется с помощью ключевого слова **show**, использование которого допускаетсяс литералами и идентификаторами.

show(<идентификатор>|<литерал>)!

Примеры применения: show(x)! show(“str”)!

* 1. Точка входа

В языке ZEI-2020 может быть только одна точка входа, определяющаяся функцией perform. В случае её отсутствия или наличия более одного её экземпляра будет сгенерировано и записано в протокол исключение. Работа транслятора будет прервана.

Синтаксис определения точки входа представлен в таблице 1.6.

* 1. Препроцессор

Препроцессоры в языке ZEI-2020 не предусмотрены.

* 1. Объектный код

Язык ZEI-2020 транслируется в язык C#.

* 1. Классификация сообщений транслятора

Генерируемые транслятором сообщения определяют степень его информативности, то есть сообщения транслятора должны давать максимально полную информацию о допущенной пользователем ошибке при написании программы. Сообщения транслятора приведены в таблице 1.8.

Таблица 1.8 Классификация ошибок.

|  |  |  |
| --- | --- | --- |
| Номера ошибок | Характеристика | Префикс |
| 0 – 199 | Системные ошибки | [!] |
| 300 – 399 | Ошибки лексического анализа | [SA] |
| 500 – 599 | Ошибки синтаксического анализа | [SX] |
| 600 – 699 | Ошибки семантического анализа | [SM] |
| 200-299, 400-499 | Зарезервированные коды ошибок |  |

1.23 Контрольный пример

Контрольный пример, демонстрирующий главные особенности языка ZEI-2020, представлен в приложении А.

# 2 Структура транслятора

## Компоненты транслятора, их назначение и принципы взаимодействия

Транслятор - программа, которая преобразует исходный код на одном языке в исходный код на другом языке программирования.

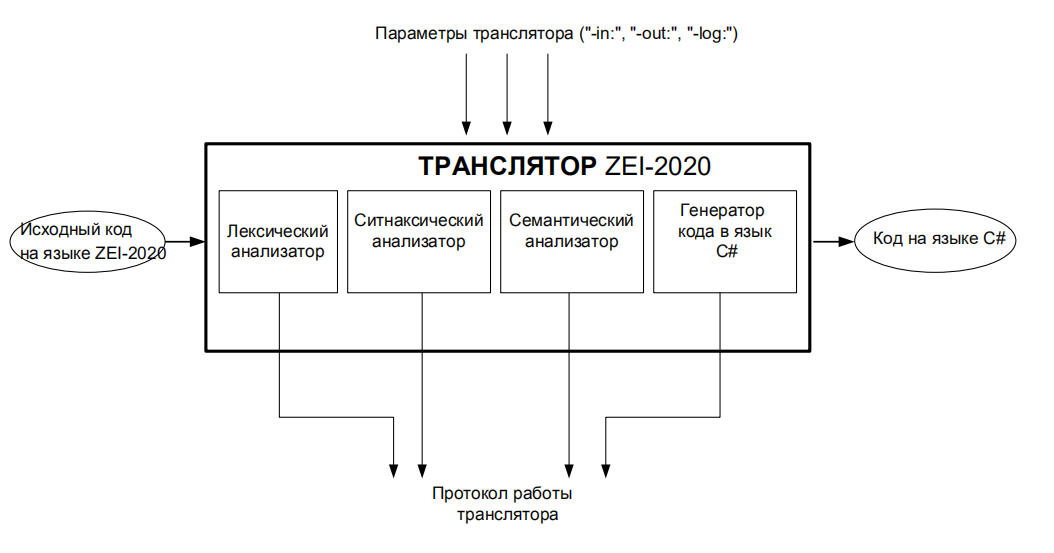
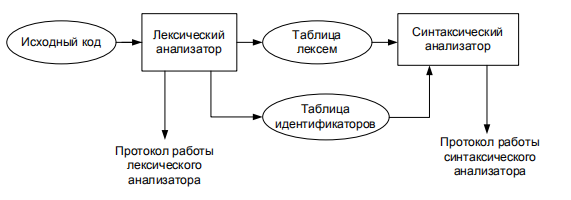
Основными компонентами транслятора ZEI-2020 являются лексический анализатор, синтаксический анализатор, семантический анализатор и генератор кода в язык C#. Они приведены на рисунке 2.1.

Рисунок 2.1 Схема структуры транслятора.

Таблица 2.1 Структура транслятора.

|  |  |
| --- | --- |
| Компонент | Предназначение |
| 1 | 2 |
| Лексический анализатор | Обрабатывает входной файл исходного кода, проверяя допустимость его символов. Удаляет избыточные символы (пробелы, символы табуляции). Выделяет простейшие конструкции языка(лексические единицы). Лексический анализатор преобразует исходный текст, заменяя лексические единицы их внутренним представлением – лексемами, для создания промежуточного представления исходной программы. Результатом работы лексического анализатора являются таблица лексем и таблица идентификаторов. |
| 1 | 2 |
| Синтаксический анализатор | Взаимодействует с лексическим анализатором последовательно (рисунок 2.2). Входом для синтаксического анализа является таблица лексем и таблица идентификаторов. Он проверяет правильность написанных конструкций, выявляет синтаксические ошибки. Результатом работы синтаксического анализатора является дерево разбора выражения. |
| Семантический анализатор | Семантический анализатор выявляет ошибки семантики (смысловые). Его входом являются таблица лексем и идентификаторов. |
| Генератор кода | Входом генератора кода является таблица лексем и таблица идентификаторов. На их основе он выполняет генерацию кода на языке C#. |

Рисунок 2.1 Последовательное взаимодействие лексического и синтаксического анализаторов.

## Перечень входных параметров транслятора

Перечень входных параметров транслятора языка ZEI-2020 представлен в таблице 2.2.

Таблица 2.2 Входные параметры транслятора.

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Тип параметра |
| 1 | 2 | 3 |
| -in: | указывает транслятору путь к исходному коду, находящемуся в файле с расширением «.txt»; | обязательный; |
| 1 | 2 | 3 |
| -log: | указывает транслятору путь к файлу протокола; | не обязательный (при отсутствии явного указания путь к файлу протокола формируется конкатенацией имени файла исходного кода с постфиксом «.log»); |
| -out: | указывает транслятору выходной файл; | не обязательный (при отсутствии явного указания конкатенацией имени файла исходного кода с постфиксом «.cs»). |

## Перечень протоколов, формируемых транслятором и их содержимое

По итогам своей работы транслятор формирует один протокол, согласно заданным входным параметрам. -log: <путь к файлу>.

Информация, записываемая в протокол, представлена в таблице 2.3.

Таблица 2.3 Протокол транслятора.

|  |  |
| --- | --- |
| Тип информации | Описание информации |
| дата и время | выводится дата и время создания протокола. |
| параметры командой строки | выводится информация об указанных параметрах командной строки. |
| информация об исходном коде | общее количество символов, количество проигнорированных символов, количество строк исходного файла. |
| таблица лексем | выводится таблица лексем. |
| таблица идентификаторов | выводится таблица идентификаторов, указывающая номер строки в таблице лексем, тип идентификатора, тип данных идентификатора, его имя, видимость, значение. |
| трассировочная информация синтаксического анализа | выводится полная информация о разборе таблицы лексем синтаксическим анализатором. |
| дерево разбора | в случае успешного разбора выводятся правила, по которым осуществился разбор исходного кода. |

# 3 Разработка лексического анализатора

## Структура лексического анализатора

Определение и назначение лексического анализатора было описано в пункте 2.1.

Структура лексического анализатора ZEI-2020 представлена на рисунке 3.1.

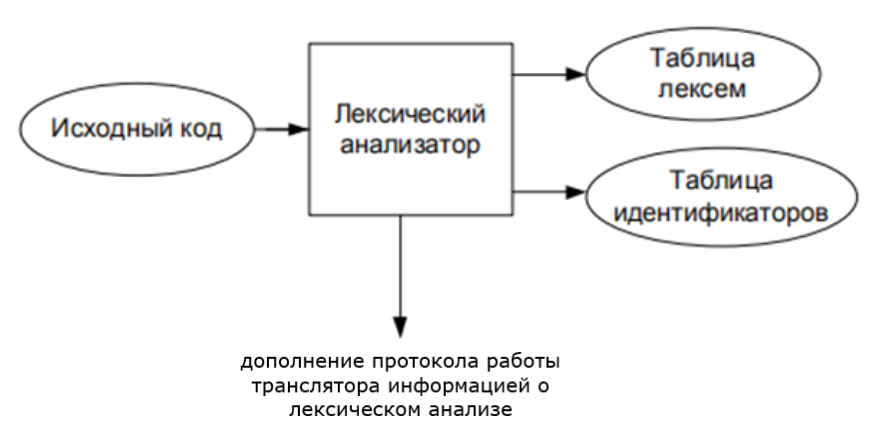


Рисунок 3.1 Структура лексического анализатора.

Исходный код на языке ZEI-2020 является входными данными лексического анализатор, выходными же являются таблицы лексем и идентификаторов.

* 1. **Контроль входных символов**

Таблица для контроля входных символов представлена на рисунке 3.2.

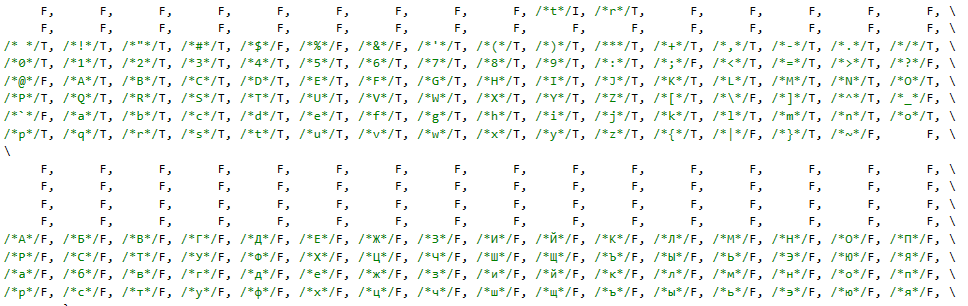


Рисунок 3.2 Таблица допустимости входных символов.

Каждому элементу таблицы соответствует тот же код, что и в таблице кодировки Windows-1251 (рисунок 1.1).

В представленной таблице F — запрещённый символ, T — разрешённый символ, I — игнорируемый символ. Каждому символу из файла с исходным кодом ставится в соответствие одно из этих значений. В случае запрещённого символа, при условии, что он встречается не в комментарии и не в строковом литерале, работа транслятора завершается, генерируется и записывается в протокол исключение. Символ, отмеченный как I, будет проигнорирован. Разрешенный символ будет допущен.

## Удаление избыточных символов

Избыточными символами являются символы табуляции и пробелы. Они удаляются сразу после получения исходного текста программы.

Перед разбиением кода на лексемы запускается функция, которая удаляет повторяющиеся пробельные символы, а также все пробельные символы вокруг знаков операций (арифметических, логических, сдвигов) и вокруг символов сепараторов. При этом, если избыточный символ находится в строковом литерале, его удаление не происходит.

Примеры одного фрагмента кода до и после удаления избыточных символов представлены на рисунках 3.3 и 3.4.

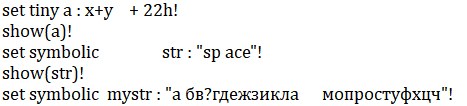


Рисунок 3.3 Фрагмент кода до удаления избыточных символов.

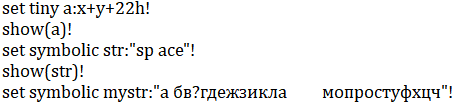


Рисунок 1.4 Фрагмент кода после удаления избыточных символов.

## Перечень ключевых слов, сепараторов, символов операций, соответствующим им лексемам и конечных автоматов

Перечень ключевых слов и соответствующих им лексем представлен в таблице 3.1.

Таблица 3.1 Перечень ключевых слов и соответствующих им лексем.

|  |  |  |
| --- | --- | --- |
| Слово | Лексема | Пояснение |
| 1 | 2 | 3 |
| set | s | объявление переменной |
| tiny, symbolic, logical | t | типы данных |
| 1 | 2 | 3 |
| when | w | условный оператор, блок, выполняющийся при истинности условия |
| otherwise | o | условный оператор, блок, выполняющийся при ложности условия |
| show | h | вывод данных |
| function | f | определение функции |
| giveback | g | возвращение значения из функции |
| perform | p | точка входа в программу |
| loop | y | цикл |
|  | i | идентификатор |
|  | l | литерал |
|  | b | библиотечная функция |

Лексемы символов + , - \* # = ^ < > [ ] ( ) < > , : \ / соответствуют самим же символам.

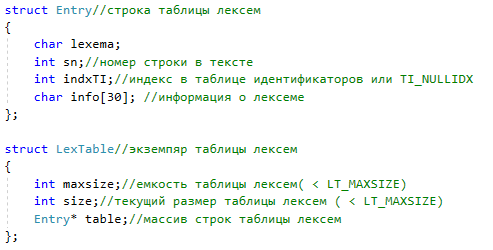
Также присутствуют лексемы false и true для литерала типа logical.

Для всех лексем определены конечные автоматы, по которым происходит разбор выражения: на каждый автомат в массиве подаётся фраза и с помощью графа переходов происходит разбор. Если разбор выполнен, происходит заполнение таблицы лексем и, при необходимости, таблицы идентификаторов.

Благодаря замене цепочек, написанных на языке ZEI-202, лексемами, упрощается дальнейшая обработка исходного кода программы. Перечень выражений представлен в приложении Б.

* 1. **Основные структуры данных**

Описание основных структур данных, используемых для хранения таблиц лексем, представлено на рисунке 3.5.

Рисунок 3.5 Структуры для таблицы лексем ZEI-2020.

Описание основных структур данных, используемых для хранения таблиц идентификаторов, представлено в приложении В.

## Принцип обработки ошибок

При обнаружении ошибки на этапе лексического анализа немедленно генерируется исключение, содержащее следующую информацию: код ошибки, номер строки в коде номер столбца в коде. При возникновении ошибки работа транслятора прекращается, а в протокол заносится информация об ошибке.

## Структура и перечень сообщений лексического анализатора

Сообщения лексического анализа помечены префиксом [LA]. Их перечень представлен в таблице 3.2.

Таблица 3.2 Перечень сообщений лексического анализатора.

|  |  |
| --- | --- |
| Код | Сообщение |
| 1 | 2 |
| 300 | Превышен максимальный размер таблицы лексем |
| 301 | Таблица лексем переполнена |
| 302 | Таблица лексем, вероятно, ещё не создана |
| 303 | Недопустимый номер строки таблицы лексем |
| 304 | Превышен максимальный размер таблицы идентификаторов |
| 305 | Таблица идентификаторов переполнена |
| 306 | Таблица идентификаторов, вероятно, ещё не создана |
| 1 | 2 |
| 307 | Недопустимый номер строки таблицы идентификаторов |
| 308 | Превышена максимальная длина имени идентификатора |
| 309 | Значение вне диапазона для литерала типа tiny [-128; +127] |
| 310 | Превышена максимальная длина литерала типа symbolic |
| 311 | Ошибка лексического разбора |

## Параметры лексического анализатора и режимы его работы

Исходный текст на языке ZEI-2020 подается на вход. Параметры, определяющие режим работы лексического анализатора, не предусмотрены.

## Алгоритм лексического анализа

Лексический анализ является первой и наиболее простой фазой трансляции. Алгоритм лексического анализатора заключается в следующем: после разбиения текста из файла с исходным кодом на слова, для каждого слова подбирается конечный автомат, способный его разобрать, в случае, если такой автомат существует, цепочка будет разобрана, иначе будет сгенерировано исключение. Далее сканер анализирует лексему, соответствующую данному слову, и выполняет действия, описанные для данной лексемы. Лексический анализатор продолжает работать, пока не будет разобрано последнее слово.

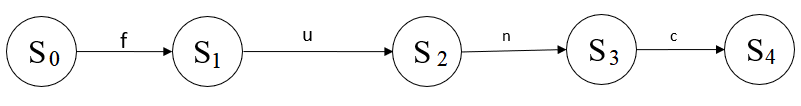
Работу конечных автоматов можно представить в виде графа. Пример графа представлен на рисунке 3.5. В виде кода представлен на рисунке 3.6. На рисунке 3.5 осуществляется разбор цепочки “func”, где S0 — начальное состояние, а S4 — конечное.

Рисунок 3.5 Граф переходов для цепочки “func”, графическое представление.

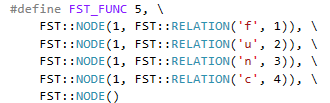


Рисунок 3.6 Граф переходов для цепочки “func”.

## Контрольный пример

Результатом работы лексического анализатора являются таблица лексем и таблица идентификаторов. Содержимое таблиц на основе исходного кода из приложения А представлено в приложении Г.

# 4 Разработка синтаксического анализатора

## Структура синтаксического анализатора

Структура синтаксического анализатора представлена на рисунке 4.1.



Рисунок 4.1 Структура синтаксического анализатора ZEI-2020.

Таблицы лексем и идентификаторов являются входными данными, а дерево разбора, выводимое в файл протокола, является выходными данными синтаксического анализатора.

## Контекстно-свободная грамматика, описывающая синтаксис языка

Грамматика для синтаксического разбора языка ZEI-2020 представляется четверкой G = <T, N, P, S>, где Т — множество терминальных символов, N — множество нетерминальных символов, P — множество правил языка, S — начальный символ грамматики, являющийся нетерминалом.

В грамматике языка ZEI-2020 множество нетерминальных символов представлено следующим образом:

* S порождает правила, описывающие общую структуру программы;
* N порождает правила, описывающие основные конструкции языка;
* Е порождает правила, описывающие выражения;
* F порождает правила, описывающие список формальных параметров функций;
* P порождает правила, описывающие формальные параметры функции;
* M порождает правила, описывающие арифметические действия;
* H порождает правила, описывающие сдвиговые операции;
* K порождает правила, описывающие вызов функции;
* W порождает правила, описывающие фактические параметры функции;
* B порождает правила, описывающие структуру тела функции;
* I порождает правила, описывающие литерал или идентификатор;
* R порождает правила, описывающие конструкции, допустимые условии циклов или условных конструкций;
* X порождает правила, описывающие конструкции, допустимые в теле циклов и условных операторов;

Перечень нетерминалов и порождаемых ими цепочек правил представлен в таблице 4.1.

Таблица 4.1 Правила грамматики ZEI-2020.

|  |  |
| --- | --- |
| Нетерминальный символ | Цепочки правил |
| S | tfiFBS | p[N] |
| F | () | (P) |
| P | ti, P | ti |
| B | [gI!] | [NgI!] |
| I | l | i |
| N | sti:E!N | sti!N | i:E!N | i:H!N | y(R)[X]N | h(I)!N | w(R)[X]N |  w(R)[X]o[X]N | bK!N | iK!N | sti:E! | sti! | i:E! | i:H! | y(R)[X] | h(I)! | w(R)[X] | w(R)[X]o[X] | bK! | iK! | |
| R | i<I | l<I | i>I | l > I | i=I | l=I | i^I | l^I |
| H | i/l | i\l | l/l | l\l |
| E | bK | i | l | (E) | iK | iM | lM | (E)M | iKM | bKM |
| K | () | (W) |
| W | i | l | i, W | l, W |
| M | +EM | -EM | \*EM | # EM | +E | \*E | #E | -E |
| X | i:I!X | i:E!X | i:H!X | h(I)!X | bK!X | iK!X | i:I! | i:E! | i:H!X | h(I)! | bK! | iK! |

## Построение конечного магазинного автомата

Принцип действия конечного магазинного автомата представлен на рисунке 4.2.



Рисунок 4.2 Схема автомата с магазинной памятью.

Формальное описание МП-автомата:



 - множество состояний;

 - алфавит входных символов;

 - специальный алфавит магазинных символов;

-функция переходов автомата , где  - множество подмножеств ;

 - начальное состояние автомата;

- начальное состояние магазина (маркер дна);

- множество конечных состояний.

Конфигурация (текущее состояние автомата) описывается тройкой , где - текущее состояние автомата, - остаток цепочки,  - цепочка-содержимое магазина.

Начальное состояние , - начальное состояние автомата,  - входная цепочка, - маркер дна магазина.

Цепочка  является допустимой (распознается) автоматом , если  и .

Работа автомата 

1. состояние автомата 
2. читает символ  находящийся под головкой (сдвигает ленту);
3. не читает ничего (читает , не сдвигает ленту);
4. из  определяет новое состояние , если  или .
5. читает верхний (в стеке) символ  и записывает цепочку  т.к. , при этом, если , то верхний символ магазина просто удаляется.

работа автомата заканчивается .

## Основные структуры данных

Основные структуры данных синтаксического анализатора включают в себя структуру магазинного автомата и структуру грамматики Грейбах, описывающую правила языка ZEI-2020. Правила грамматики описаны в приложении Д.

## Описание алгоритма синтаксического разбора

Входные символы и лексемы в форме Грейбах находятся в ленте на входе конечного автомата.

1) Если лента не пустая, переходим далее следующему пункту, иначе переходим к пункту 5.

2) Если на верхушке магазина нетерминальный символ.

2.1) Если есть такое правило, то переходим к следующему пункту.

2.1.1) Если цепочка есть, возвращаем NS\_OK. Переходим к пункту 4.

2.1.2) Иначе восстанавливаем состояние. Переходим к пункту 4.

2.2) Иначе возвращаем ошибку. Переход к пункту 4.

3) Если на верхушке терминал и он совпадает с символом на ленте, то удаляем его из стека и продвигаем ленту. Переход к пункту 4.

4) Повторяем шаг, переходим к пункту 1.

5) Конец работы.

## Структура и перечень сообщений синтаксического анализатора

Сообщения синтаксического анализатора отмечены префиксом [SX], и их перечень приведен в таблице 4.2.

Таблица 4.2 Перечень сообщений синтаксического анализатора.

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 500 | Неверная структура программы |
| 501 | Неверный список параметров функции |
| 502 | Неверный список параметров функции при её объявлении |
| 503 | Отсутствует тело функции |
| 504 | Неверное выражение. Ожидаются только идентификаторы и литералы |
| 506 | Неверная конструкция в теле функции |
| 507 | Ошибка в условном выражении |
| 508 | Ошибка в вызове функции |
| 509 | Ошибка в параметрах вызываемой функции |
| 510 | Ошибка в арифметических или сдвиговых операциях |
| 511 | Неверная конструкция в теле цикла / условного выражения |

## Параметры синтаксического анализатора и режимы его работы

Входных параметров для синтаксического анализатора, определяющих режим его работы, не предусмотрено.

## Принцип обработки ошибок

Принцип обработки ошибок заключается в том, что синтаксический анализатор перебирает все возможные правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем. В случае, если не была найдена ни одна подходящая цепочка, формируется соответствующая ошибка из таблицы 4.2. Все ошибки записываются в общую структуру ошибок (в которой запоминается до 3 ошибок), выводятся в файл протокола и отображаются на консоли.

## Контрольный пример

Результатом работы синтаксического анализатора для контрольного примера, представленного на рисунке 4.3, являются трассировка, представленная в приложении Е и дерево разбора выражения, представленное на рисунке 4.4 и в графическом материале.

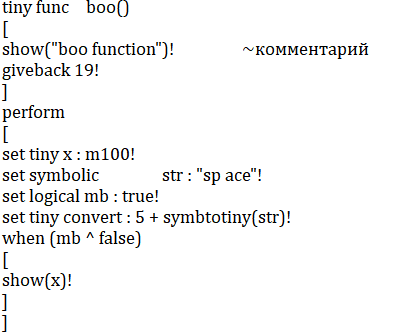


Рисунок 4.3 Контрольный пример для синтаксического анализа.

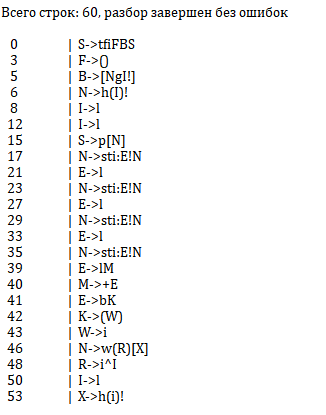


Рисунок 4.4 Дерево разбора для контрольного примера с рисунка 4.3.

# 5 Разработка семантического анализатора

## Структура семантического анализатора

Семантический анализ в трансляторе языка ZEI-2020 выделен в отдельную фазу, а также частично реализован на этапе лексического анализа. Состоит из набора инструкций для проверки смысловой корректности исходной кода.

Структура семантического анализатора представлена на рисунке 2.1.

## Функции семантического анализа

Семантические проверки языка ZEI-2020 представлены в главе 1, пункте 1.16.

Из них на этапе лексического анализа выполняются следующие:

* проверка наличия и единственности точки входа в программу perform;
* проверка на необъвленный идентификатор;
* проверка переобъявления идентификатора;

Остальные семантические проверки реализованы функцией Analysis из пространства имен Sem.

## Структура и перечень сообщений семантического анализатора

Сообщения синтаксического анализатора отмечены префиксом [SM].

Сообщения, формируемые семантическим анализатором представлены в таблице 5.1.

Таблица 5.1 Перечень сообщений семантического анализатора.

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 1 | 2 |
| 602 | Отсутствует точка входа в программу perform |
| 603 | Присутствует более одной точки входа в программу perform |
| 604 | Идентификатор не объявлен |
| 605 | Невозможно вызвать функцию с объявляемым идентификатором |
| 606 | Идентификатор уже определен |
| 607 | Неверное место для объявления функции! Должна быть определена до точки входа в программу perform |
| 608 | Несовпадение типа функции и типа возвращаемого значения |
| 609 | Превышено максимальное значение в параметрах определяемой функции (3) |
| 610 | Неверные параметры вызываемой функции |
| 611 | Слишком много аргументов в вызове функции |
| 612 | Слишком мало аргументов в вызове функции |
| 1 | 2 |
| 613 | Типы данных в выражении не совпадают |
| 614 | Арифметические операции можно производить только над типом tiny |
| 615 | Логические операции < > можно производить только над типом tiny |
| 616 | Логические операции = ^ можно производить только над типами tiny или logical |
| 617 | Неверное выражение. Несовпадение типов слева и справа от знака присваивания |

## 5.4 Принцип обработки ошибок

Все семантические ошибки являются критическими, и при генерации одной из них генерируется исключение, транслятор прекращает работу, и в протокол работы выводится соответствующее сообщение об ошибке.

## 5.5 Контрольный пример

Контрольный пример заключается в тестировании семантического анализатора при наличии соответствующих ошибок в исходном коде. Тестирование представлено в таблице 5.3.

Таблица 5.3 Тестирование.

|  |  |
| --- | --- |
| Исходный код с ошибкой | Генерируемое сообщение об ошибке |
| 1 | 2 |
| perform  [  ]  perform  [ | Ошибка 603: [SM] Присутствует более одной точки входа в программу perform |
| perform  [  show(x)! | Ошибка 604: [SM] Идентификатор не объявлен  строка 15 позиция 6 |
| perform  [  set tiny x : m100!  set tiny x : 100! | Ошибка 606: [SM] Идентификатор уже определен  строка 16 позиция 10 |
| 1 | 2 |
| perform  [  set symbolic x : m100! | Ошибка 613: [SM] Типы данных в выражении не совпадают  строка 15 позиция 0 |
| tiny func boo()  [  show("boo function")!  giveback 19!  ]  perform  [  set tiny x : m100!  boo(x)! | Ошибка 611: [SM] Слишком много аргументов в вызове функции  строка 16 позиция 0 |
| set logical mb : true!  set tiny convert: symbtotiny(mb)! | Ошибка 610: [SM] Неверные параметры вызываемой функции  строка 29 позиция 0 |

# 6 Вычисление выражений

## Выражения, допускаемые языком

В языке ZEI-2020 допускаются выражения с использованием числовых идентификаторов и литералов, а также вызовом функций. Предусмотрены операции, описанные в пункте 1.12.

Примеры допускаемых языком выражений:

* set tiny fef : (x -12)# (a +y)!
* set tiny a : x+y + 22h!
* convert : 2 + x + symbtotiny(str)!
* mb ^ false!
* y : x \ 2!

# 7 Генерация кода

## 7.1 Структура генератора кода

Генерация кода — это перевод транслятором представления исходной программы на языке ZEI-2020 в цепочку символов выходного языка C#. На вход генератора подаются таблицы лексем и идентификаторов на основе которых генерируется файл с кодом на языке C#.

Схематично генерация кода показана на рисунке 7.1.

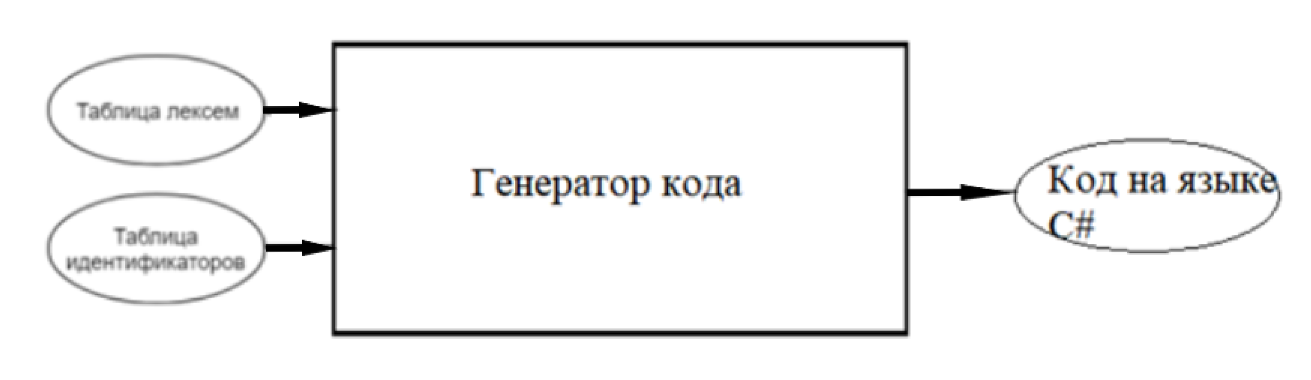


Рисунок 7.1 Структура генератора кода.

## 7.2 Представление типов данных в оперативной памяти

Язык ZEI-2020 требует указывать тип данных при объявлении идентификатора с помощью ключевого слова set. Соответствия между типами данных идентификаторов языка ZEI-2020 и языка C# представлены в таблице 7.1.

Таблица 7.1 Соответствие типов данных.

|  |  |
| --- | --- |
| Тип данных на языке ZEI-2020 | Тип данных на языке C# |
| tiny (целочисленный, 1 байт) | sbyte (целочисленный, 1 байт) |
| symbolic (строковый) | string (строковый) |
| logical (логический) | bool (логический) |

## 7.3 Библиотека

В языке ZEI-2020 предусмотрена стандартная библиотека, представленная файлом StandartLibrary.cs. Она содержит функции, написанные на языке C#, которые описаны в таблице 1.9.

Объявление функций библиотеки происходит автоматически.

Вызовы стандартных функций доступны там же, где и вызов пользовательских функций.

Фрагмент генератора кода, обеспечивающий работу библиотечных функций, приведен на рисунке 7.2.



Рисунок 7.2 Фрагмент генератора кода, обеспечивающий работу библиотечных функций.

## 7.4 Особенности алгоритма генерации кода

Одной из особенностей генерации кода является добавление явного приведения к типу sbyte в коде на языке C# в те инструкции, где находятся выражения типа tiny, применяющие арифметические операции, операции сдвига или в места возвращения из функции значения типа tiny.

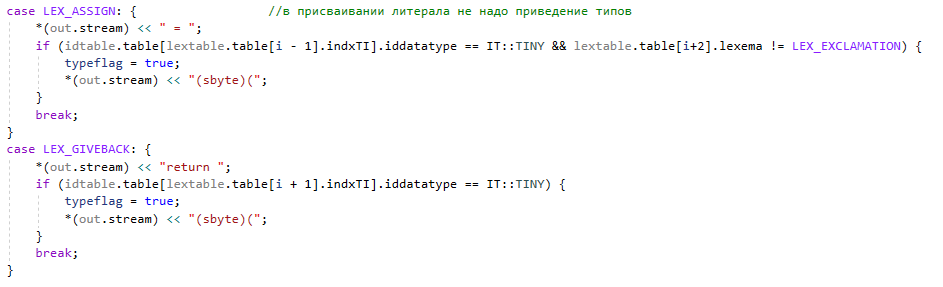


Рисунок 7.3 Фрагмент генератора кода, добавляющего явное приведение типов.

Такое приведение типов необходимо потому, что в языке C# нельзя применять многие операции (в том числе арифметические) к sbyte типу. Операции выполняются над типом System.Int32, а затем приводятся к необходимому однобайтовому типу.

Примеры подобных приведений представлены в таблице 7.2.

Таблица 7.2 Соответствие выражений при приведении типов.

|  |  |
| --- | --- |
| Выражение на языке ZEI-2020 | Выражение на языке C# |
| giveback 19! | return (sbyte)(19); |
| y : x \ 2! | y = (sbyte)(x << 2); |
| set tiny a : x+y + 22h! | sbyte a = (sbyte)(x + y + 34); |
| set tiny convert : 2 + x + symbtotiny(str)! | sbyte convert = (sbyte)(2 + x + ZEI2020stdlib.StandartLibrary.SymbToTiny(str)); |
| set tiny fef : (x -12) # (a +y)! | sbyte fef = (sbyte)((x - 12) / (a + y)); |

Также на этапе генерации кода осуществляется инициализация переменных по умолчанию, описанная в подразделе 1.5.

Еще одной особенностью является генерация отступов в исходном коде на языке C# для лучшего представления структуры программы.

Сразу после успешной генерации кода есть возможность вызвать компилятор C# (csc.exe) непосредственно из транслятора ZEI-2020. В передаваемые ему параметры входит путь к файлу, заданный ключом -out при вызове транслятора. Компилятор csc.exe формирует исполнимый файл, а затем он также вызывается непосредственно из транслятора ZEI-2020.



Рисунок 7.2 Реализация запуска программы на языке C# из транслятора ZEI-2020.

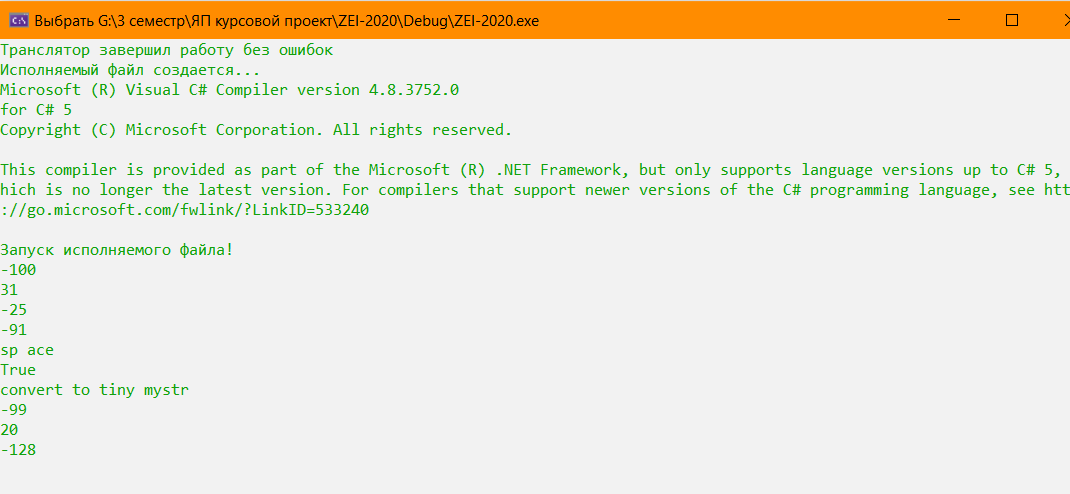


Рисунок 7.3 Демонстрация запуска программы на языке C# из транслятора ZEI-2020.

## 7.5 Входные параметры генератора кода

На этапе генерации кода входные параметры транслятора не предусмотрены.

## 7.6 Контрольный пример

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

# 8 Тестирование транслятора

## 8.1 Общие положения

В результате обработки исходного кода программы, представленного в приложении А, транслятор языка ZEI-2020 генерирует общий протокол работы, куда записываются все возникшие ошибки. Также они выводятся и на консоль.

## 8.2 Результаты тестирования

Транслятор языка ZEI-2020 представляет диагностику и выявление ошибок на разных этапах трансляции.

Тестирование ошибок транслятора представлено в приложении И.

# Заключение

В данном курсовом проекте были выполнены поставленные минимальные требования. Были успешно достигнуты поставленные цели, представленные во введении.

В итоге был разработан язык программирования ZEI-2020 и транслятор к нему.

Окончательная версия языка ZEI-2020 включает:

* 3 типа данных;
* поддержка операции вывода;
* возможность вызова функций стандартной библиотеки;
* 4 арифметических оператора для вычисления выражений;
* 4 логических оператора;
* 2 оператора сдвига;
* структурированная система для обработки ошибок пользователя.

Основные характеристики транслятора ZEI-2020:

* Возможность обработки 3 входных параметров;
* Возможность обработки 49 ошибок;
* Реализация 39 конечных автоматов;
* Реализация 78 цепочек правил грамматики;
* Наличие порядка 2900 строк кода.

# Список использованных источников

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

2. Смелов, В.В. Курс лекций по предмету языки программирования – 2016

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

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

# Приложение А

Контрольный пример на языке ZEI-2020

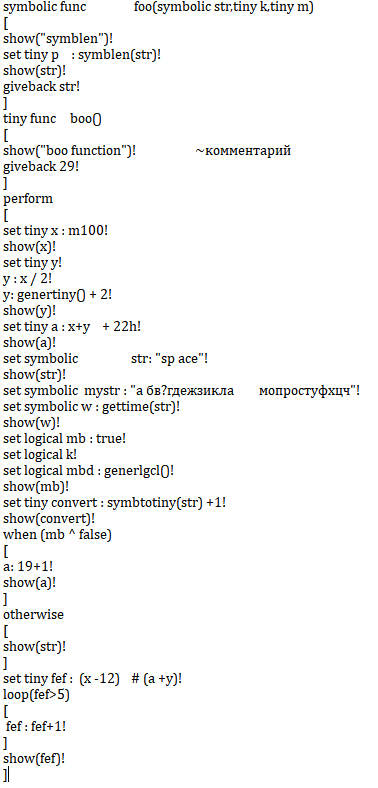


Рисунок 1 Контрольный пример.

# **Приложение Б**

Графы переходов конечных автоматов

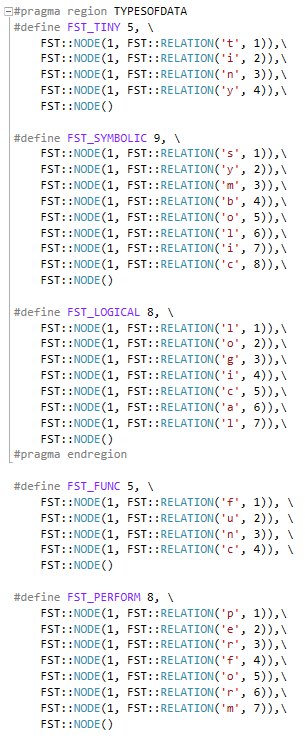
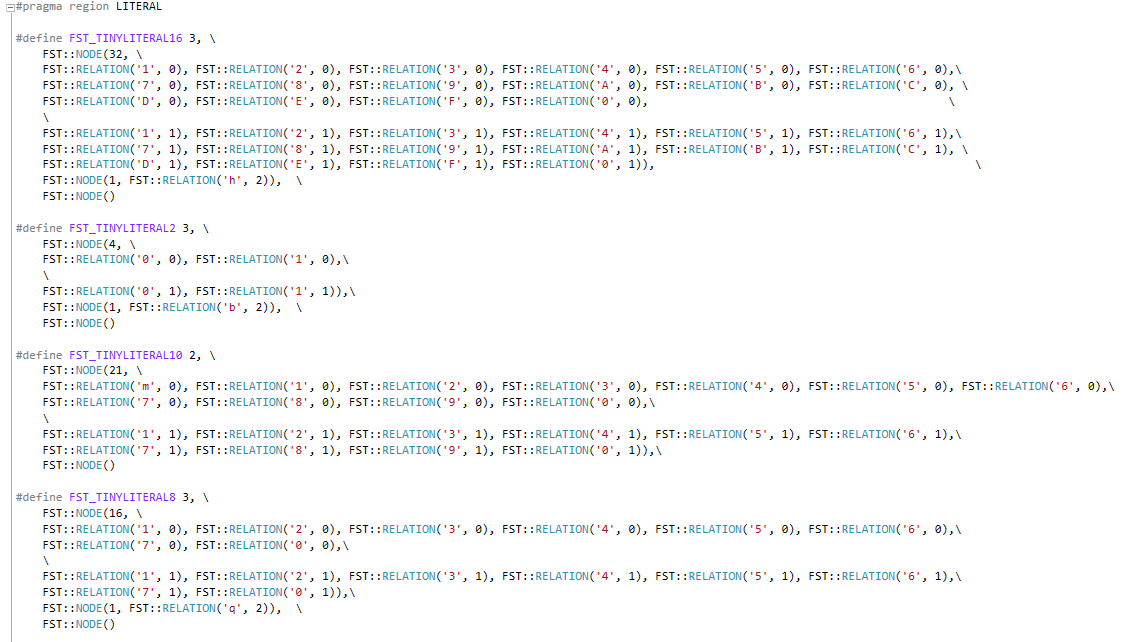


Рисунок 2 Графы для типов данных, функции и точки входа.

Продолжение приложения Б



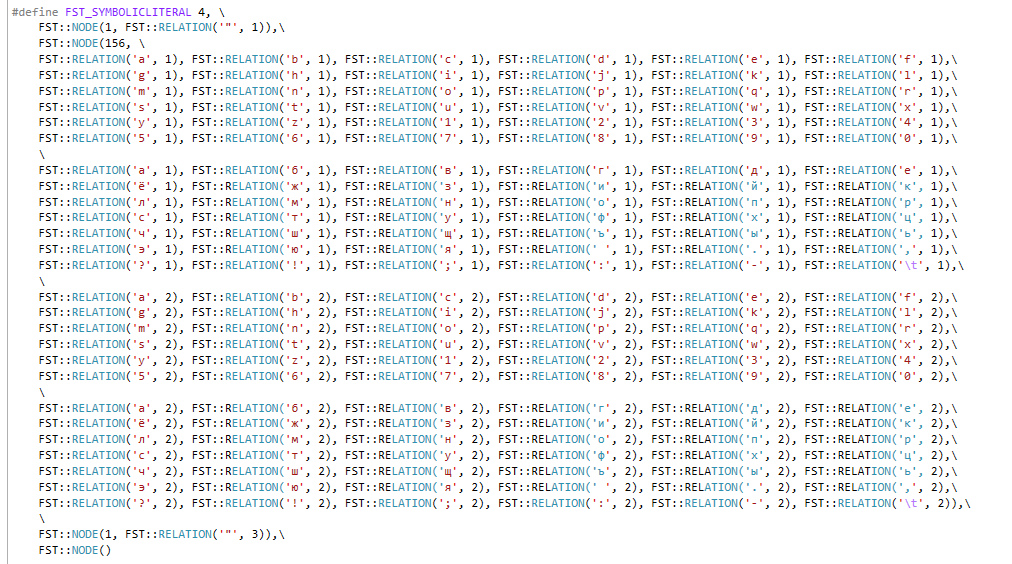


Рисунок 3 Графы для литералов.

Продолжение приложения Б

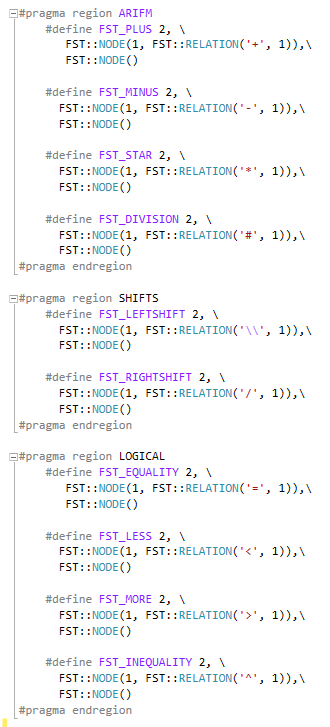


Рисунок 4 Графы арифметических и логических операторов, операторов сдвига.

# Приложение В

Структуры для представления таблицы идентификаторов

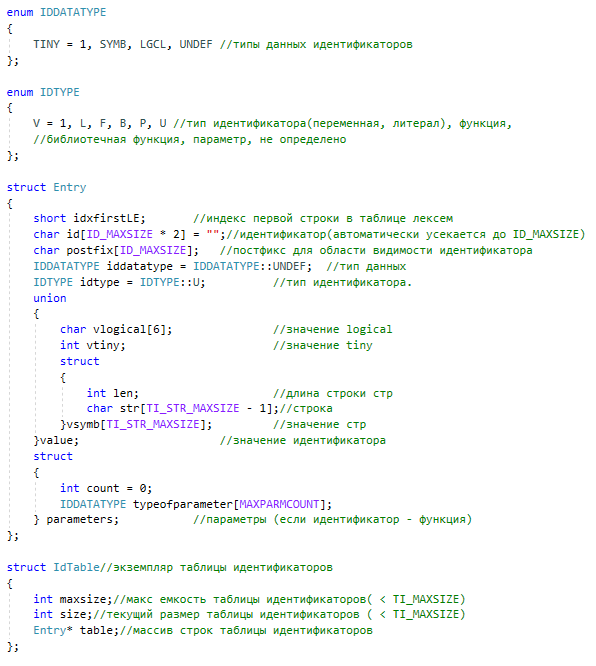


Рисунок 5 Структуры для представления таблицы идентификаторов.

# Приложение Г

Таблица лексем

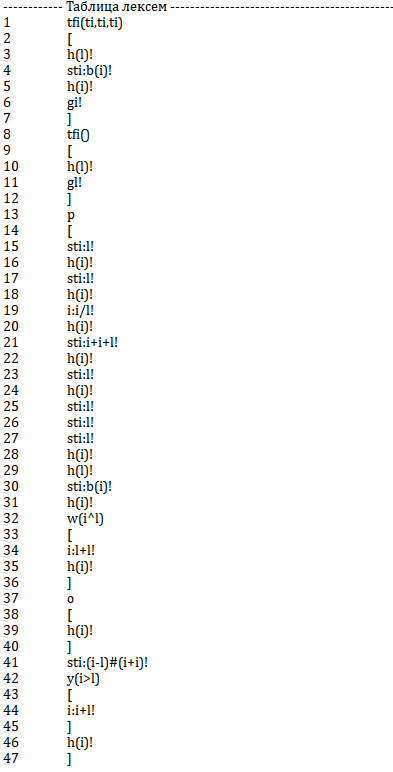


Рисунок 6 Таблица лексем.

Продолжение приложения Г

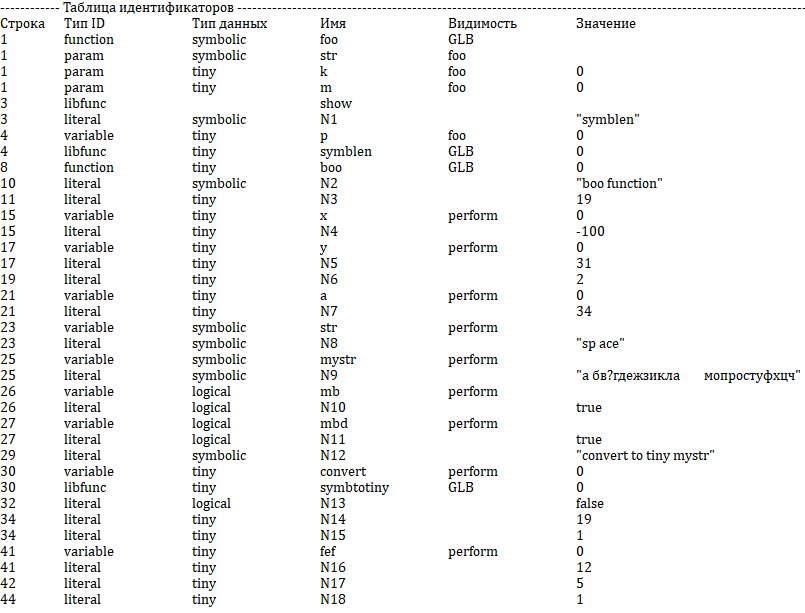
Таблица идентификаторов

Рисунок 7 Таблица идентификаторов.

# Приложение Д

Цепочки правил для синтаксического анализа

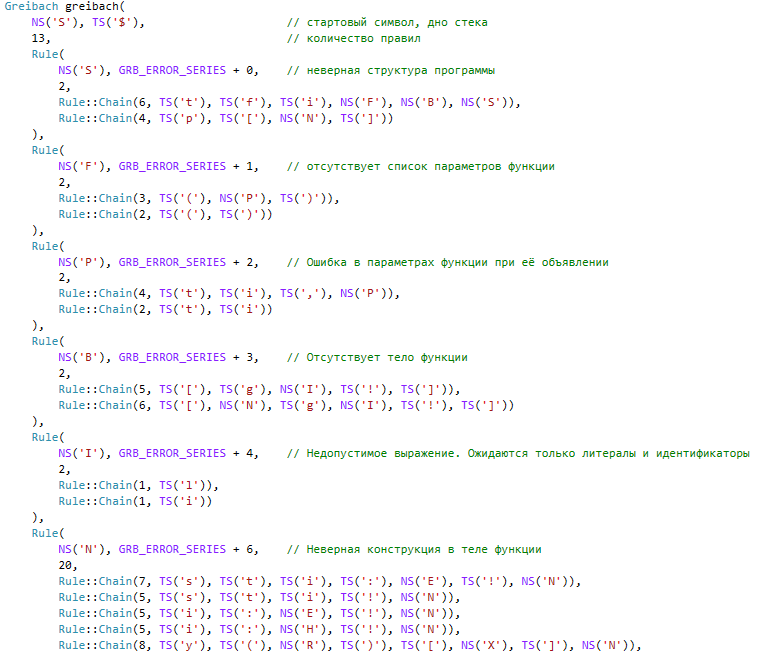
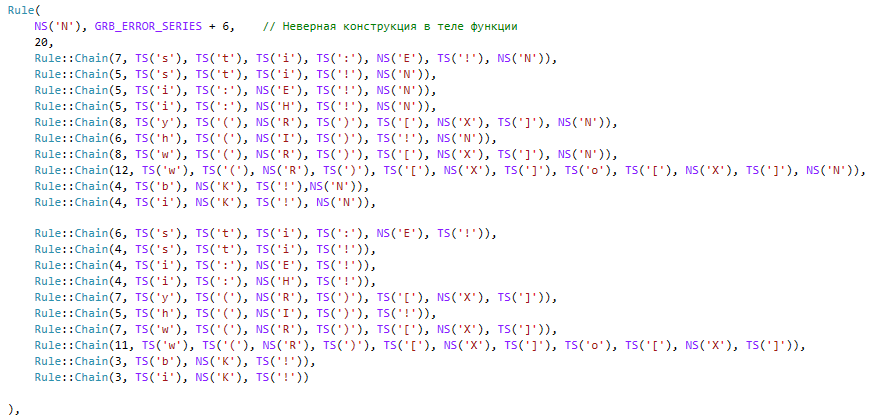


Рисунок 8 Грамматика языка, нетерминалы S, F, P, B, I, N.

Продолжение приложения Д



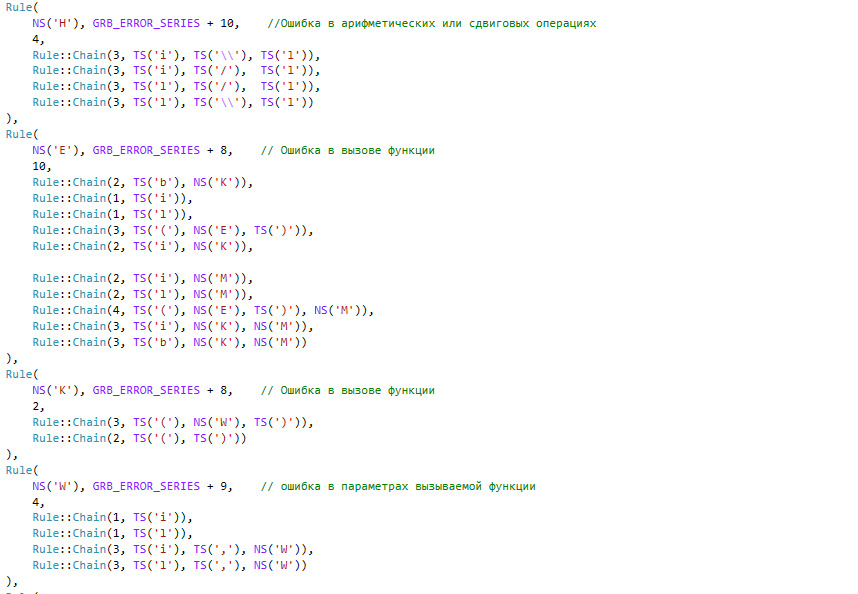
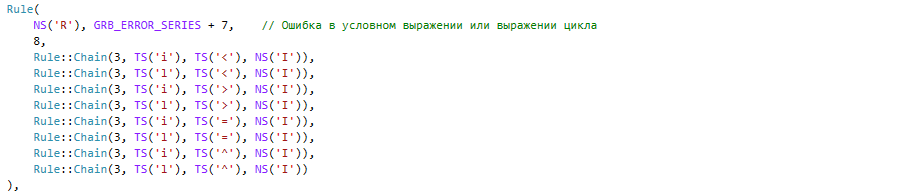
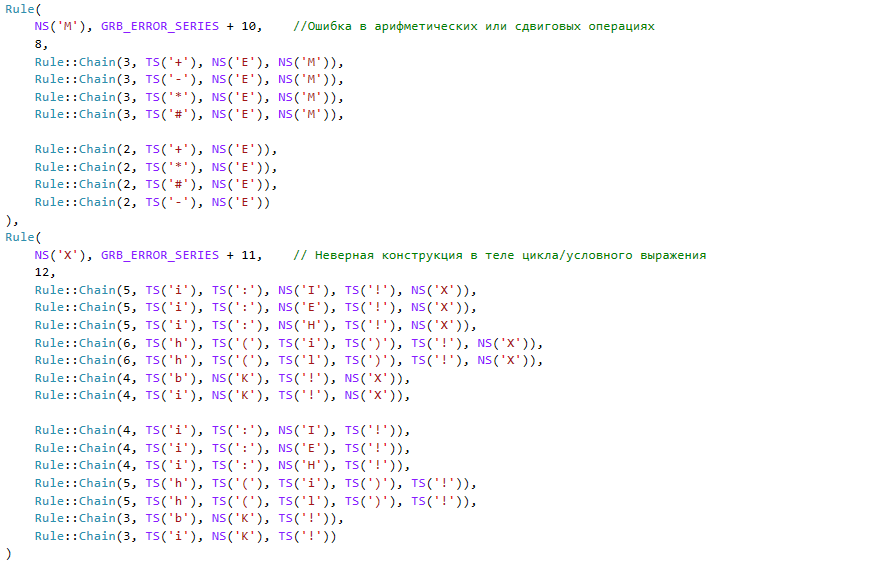


Рисунок 9 Грамматика языка, нетерминалы N, R, H, E, K, W.

Продолжение приложения Д

Рисунок 10 Грамматика языка, нетерминалы M, X.

# Приложение Е

Трассировка синтаксического разбора

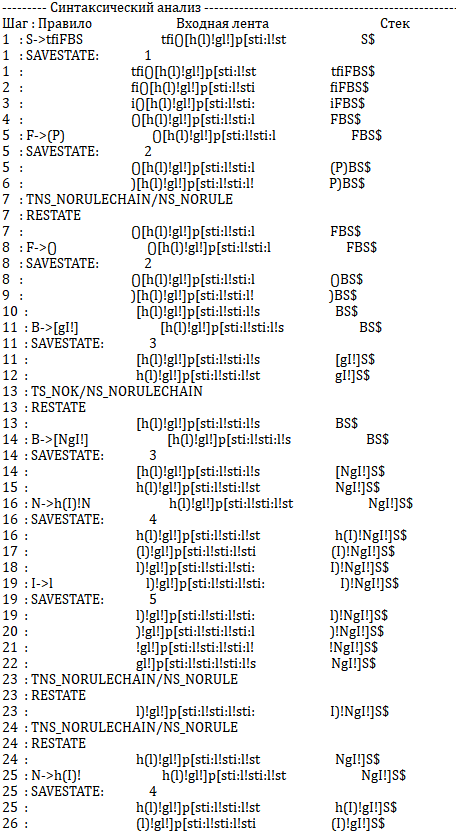


Рисунок 11 Трассировка синтаксического разбора.

Продолжение приложения Е

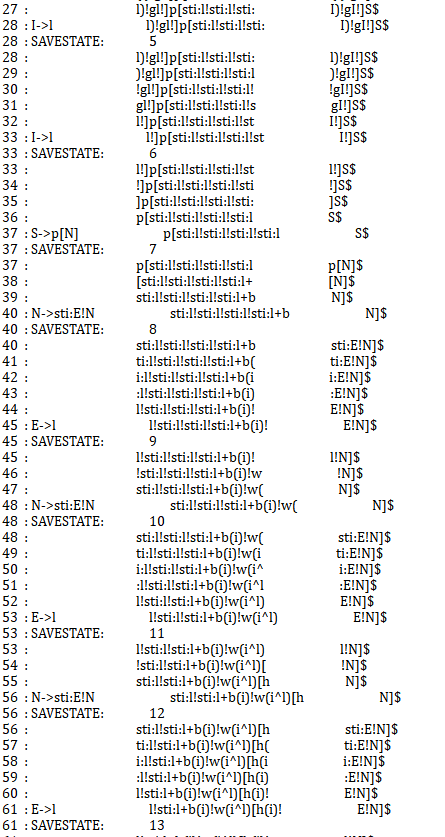


Рисунок 12 Продолжение трассировки синтаксического разбора.

Продолжение приложения Е

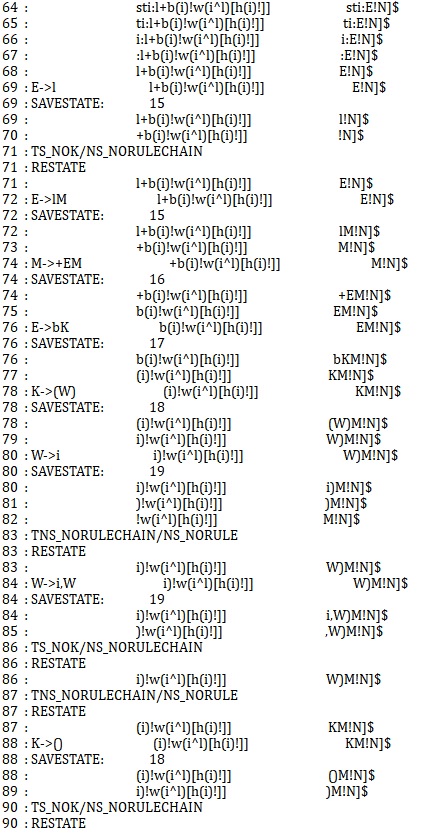


Рисунок 13 Продолжение трассировки синтаксического разбора.

Продолжение приложения Е

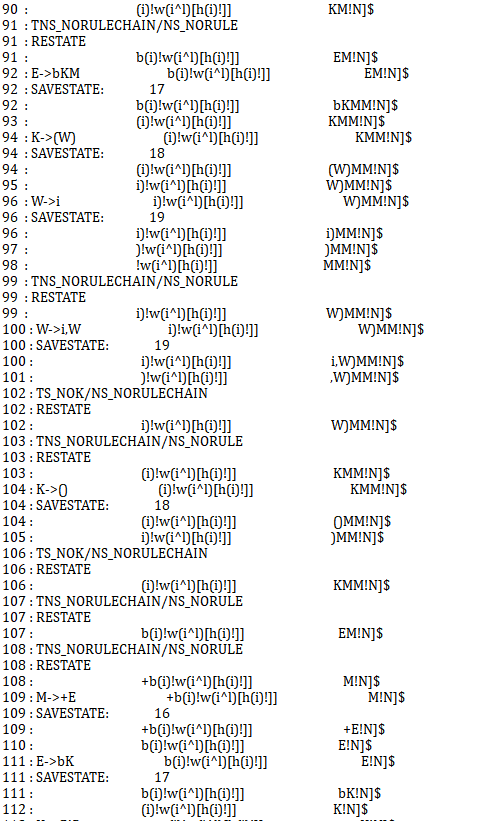


Рисунок 13 Продолжение трассировки синтаксического разбора.

Продолжение приложения Е

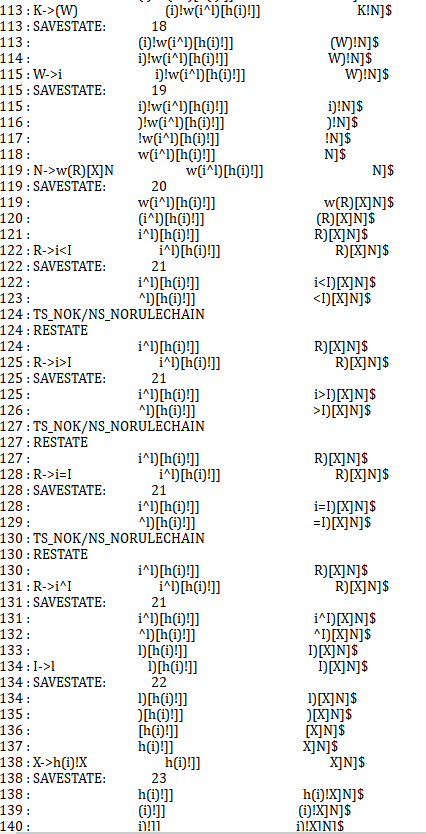


Рисунок 14 Продолжение трассировки синтаксического разбора.

Продолжение приложения Е

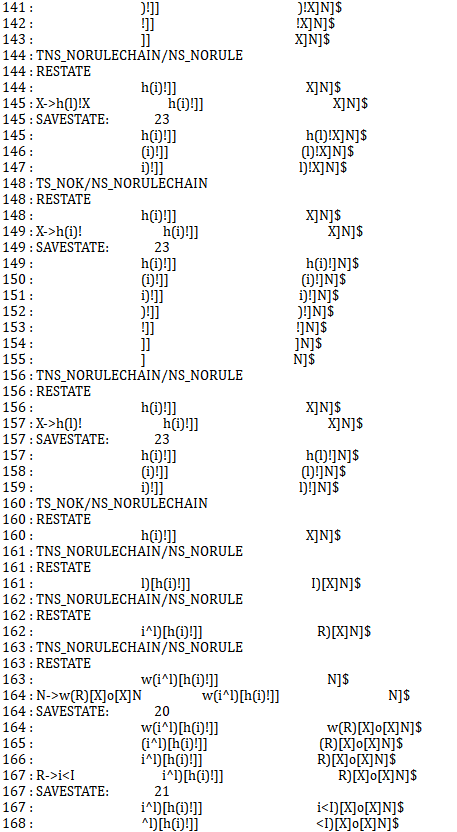


Рисунок 15 Продолжение трассировки синтаксического разбора.

Продолжение приложения Е

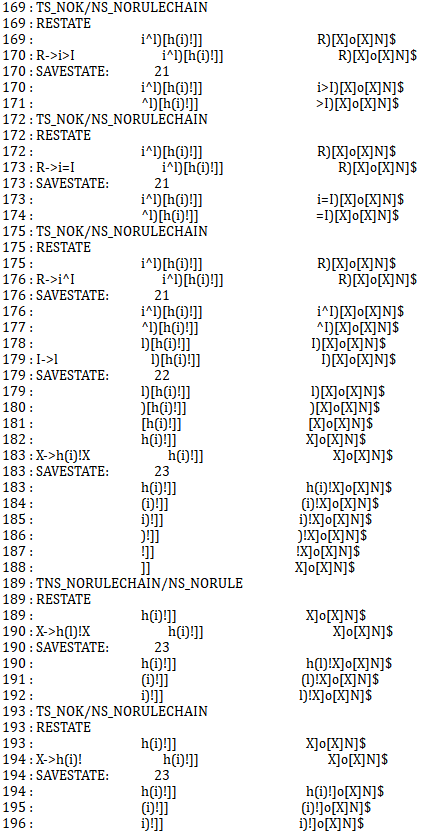


Рисунок 16 Продолжение трассировки синтаксического разбора.

Продолжение приложения Е

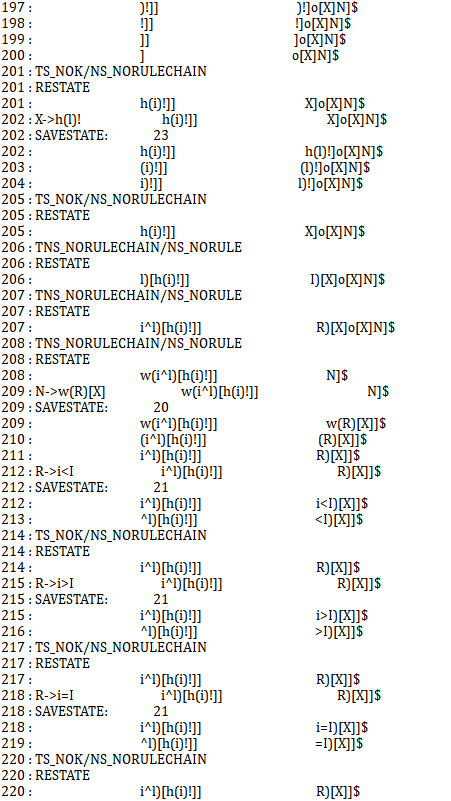


Рисунок 17 Продолжение трассировки синтаксического разбора.

Продолжение приложения Е

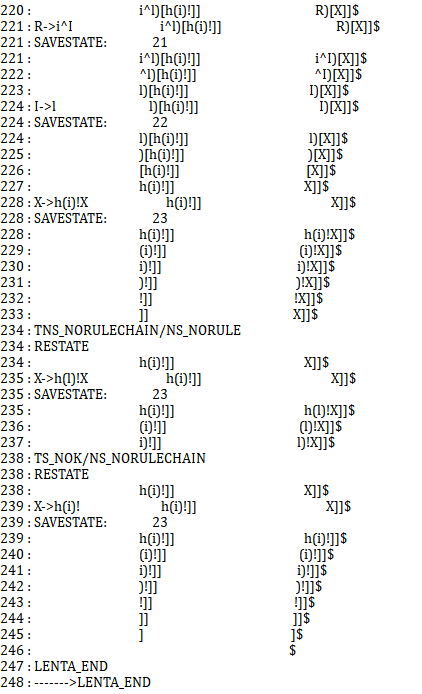


Рисунок 18 Конец трассировки синтаксического разбора.

# Приложение Ж

Результат генерации кода

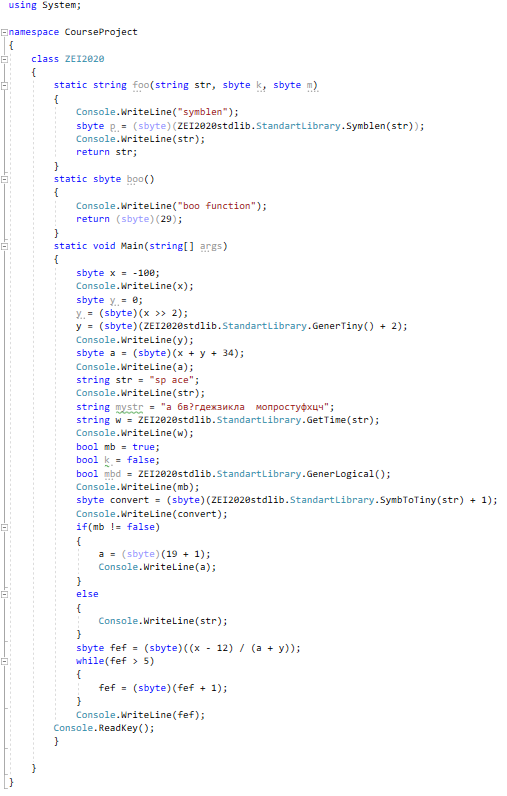


Рисунок 19 Исходный код программы на языке C#.

# Приложение И

Тестирование ошибок транслятора

Таблица 1 Тестирование.

|  |  |
| --- | --- |
| Фрагмент исходного кода | Генерируемое исключение |
| 1 | 2 |
| show(convert)!  otherwise  [  show(str)!  ] | 506: line 33, [SX] Неверная конструкция в теле функции |
| set symbolic strstrstrstrstrstrstr : "sp ace"! | Ошибка 308: [LA] Превышена максимальная длина имени идентификатора  строка 23 позиция 34 |
| set tiny y : 378! | Ошибка 309: [LA] Значение вне диапазона для литерала типа tiny [-128; +127]  строка 17 позиция 14 |
| set tiny y : 18q! | Ошибка 311: [LA] Ошибка лексического разбора  строка 17 позиция 14 |
| set tiny y : m12h! | Ошибка 311: [LA] Ошибка лексического разбора  строка 17 позиция 15 |
| set symbolic y: 3! | Ошибка 613: [SM] Типы данных в выражении не совпадают  строка 17 позиция 0 |
| tiny func boo()  [  show("boo function")!  ] | 506: line 11, [SX] Неверная конструкция в теле функции  504: line 10, [SX] Неверное выражение. Ожидаются только идентификаторы и литералы  504: line 10, [SX] Неверное выражение. Ожидаются только идентификаторы и литералы |
| 1 | 2 |
| tiny func boo()  [  show("boo function")!~комментарий  giveback "wd"!  ] | Ошибка 608: [SM] Несовпадение типа функции и типа возвращаемого значения  строка 8 позиция 0 |
| set tiny переменная : m100! | Ошибка 111: [!] Недопустимый символ в исходном файле (-in)  строка 15 позиция 11 |
| tiny func boo()  [  ] | 506: line 10, [SX] Неверная конструкция в теле функции  503: line 9, [SX] Отсутствует тело функции  501: line 8, [SX] Неверный список параметров функции |
| when (mb > false)  [  a: 19+1!  show(a)!  ] | Ошибка 615: [SM] Логические операции < > можно производить только над типом tiny  строка 32 позиция 0 |
| set logical mb : true!  set logical mbd : true + mb! | Ошибка 614: [SM] Арифметические операции можно производить только над типом tiny  строка 27 позиция 0 |
| set logical mbd : 0! | Ошибка 613: [SM] Типы данных в выражении не совпадают  строка 27 позиция 0 |
| 1 | 2 |
| symbolic func foo(symbolic str,tiny k,tiny m,logical o)  [  show("symblen")!  set tiny p : symblen(str)!  show(str)!  giveback str!  ] | Ошибка 609: [SM] Превышено максимальное значение в параметрах определяемой функции (3)  строка 1 позиция 54 |
| show(show)! | 504: line 31, [SX] Неверное выражение. Ожидаются только идентификаторы и литералы  504: line 31, [SX] Неверное выражение. Ожидаются только идентификаторы и литералы  504: line 31, [SX] Неверное выражение. Ожидаются только идентификаторы и литералы |