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 ZRV-2018»

Выполнил студент Завадский Роман Витальевич

(Ф.И.О.)

Руководитель проекта ст.пр. Наркевич Аделина Сергеевна

(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 ст.пр. Наркевич Аделина Сергеевна

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

преп. ст. Рауба Алексей Александрович

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

Нормоконтролер ст.пр. Наркевич Аделина Сергеевна

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

Курсовой проект защищен с оценкой

Минск 2018

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПЕУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий   
Кафедра программной инженерии

Утверждаю

Заведующая кафедрой

\_\_\_\_\_\_\_\_\_\_\_\_ Н.В. Пацей

подпись инициалы и фамилия

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2018г.

**ЗАДАНИЕ**

**к курсовому проектированию**

**по дисциплине** "Языки программирования"

Специальность: ПОИТ Группа: 6

Студент: Завадский Роман Витальевич

(фамилия, имя, отчество)

**1. Тема проекта** Разработка компилятора ZRV-2018

утверждена приказом по университету от «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2018 г. №

**2. Срок сдачи студентом законченного проекта:** \_\_\_ декабря 2018 г.

**3. Исходные данные к проекту:**

Разработка программы осуществляется на языке C++ (стандартизации International Standard ISO/IEC 14882:2014(E) Programming Language C++ 14) в среде разработки Microsoft Visual Studio Enterprise 2017. Операционная система под которой происходит разработка Windows 10 (64-bit). Типы данных: numb и letter. Функции стандартной библиотеки: numb length(letter) – длина строки, numb subline(numb, numb) – возведение в степень. Арифметические операции: +, -, \*. Оператор вывода в стандартный поток: escape. Поддерживается условный оператор if … next … else…

**4. Содержание расчетно-пояснительной записки (перечень подлежащих разработке вопросов):**

Введение

1) Спецификация языка программирования

2) Структура транслятора

3) Разработка лексического анализатора

4) Разработка синтаксического анализатора

5) Разработка семантического анализатора

6) Вычисление выражений

7) Генерация кода

8) Тестирование транслятора (Разработка и тестирование интерпретатора)

Приложения

Литература

**5. Перечень графического материала (с точным указанием обязательных чертежей)**

1) Граф

**6. Консультанты по проекту с указанием относящихся к ним разделов проекта**

|  |  |
| --- | --- |
| Раздел | Консультант |
| Разработка синтаксического и семантического анализатора. | Рауба А. А. |
| Генерация кода. Разработка тестовых примеров. | Рауба А. А. |
| Оформление пояснительной записки к курсовому проект. | Рауба А. А. |
|  |  |

**7. Календарный план**

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Наименование этапов курсового проекта | Срок выполнения этапов проекта | Примечание |
| 1 | Спецификация специализированного языка ZRV-2018 |  |  |
| 2 | Разработка лексического анализатора |  |  |
| 3 | Разработка синтаксического анализатора |  |  |
| 4 | Разработка семантического анализатора |  |  |
| 5 | Генерация кода |  |  |
| 6 | Тестирование компилятора |  |  |
| 7 | Оформление пояснительной записки к курсовому проект |  |  |
| 8 | Сдача проекта |  |  |

**8. Дата выдачи задания 15.09.2018**

Руководитель Рауба А.А.

(подпись)

Задание принял к исполнению Завадский Р.В.

(дата и подпись студента)

Содержание

**Введение**

1 Спецификация языка программирования

## **Характеристика языка программирования**

Язык ZRV-2018 – это компилируемый, строго типизированный, процедурный, универсальный язык, не являющийся объектно-ориентированным.

* 1. **Алфавит языка**

Алфавит Языка ZRV-2018 основан на кодировке Windows-1251, представленной в таблице на рисунке 1.1.

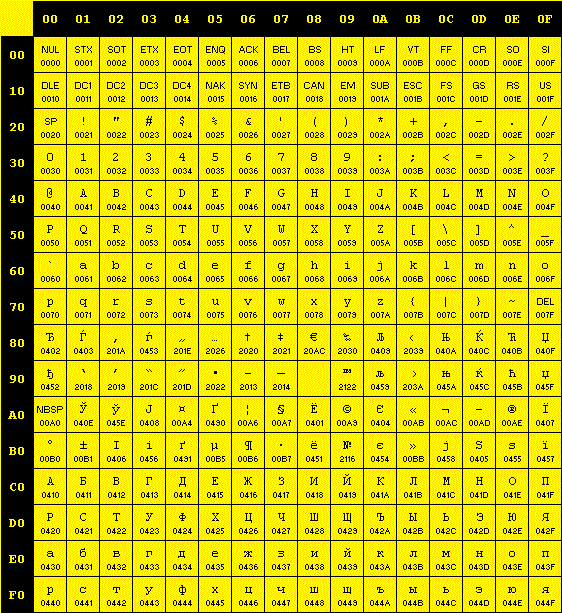


Рисунок 1.1 – Алфавит входных символов

Исходный код CHAW-2017 может содержать:

* символы латинского алфавита;
* цифры десятичной системы счисления от 0 до 9;
* русские символы разрешены только в строковых литералах.

## **Символы сепараторы**

Символы, которые являются сепараторами представлены в таблице 1.1.

Таблица 1.1 – Сепараторы языка ZRV-2018

|  |  |
| --- | --- |
| Сепаратор | Назначение |
| (…) | Блок параметров функции и приоритет операций (арифметических) |
| {…} | Программный блок |
| ‘ ‘ (пробел) | Разделитель цепочек |
| ; | Разделитель конструкций |
| = | Оператор присваивания |
| **+ - \* =** | Арифметические операции |

## **Применяемые кодировки**

Для написания исходного кода в языке ZRV-2018 используется кодировка Windows-1251.

## **Типы данных**

В языке ZRV-2018 используется 2 типа данных: целочисленный и строковый. Описание типов данных представлено в таблице 1.2.

Таблица 1.2 – Типы данных языка ZRV-2018

|  |  |
| --- | --- |
| Тип данных | Описание |
| numb | Фундаментальный тип данных. Объявление целочисленных положительных данных. Значение по умолчанию: 0. В памяти занимает 4 байта. Операции:  + - бинарный, оператор сложения;  - - бинарный, оператор вычитания;  \* - бинарный, оператор умножения.  = - присваивание значения. |
| letter | Фундаментальный тип данных. Объявление строк. Строка нулевой длины " " по умолчанию. Максимальное количество символов в строке: 255. |

## **Преобразование типов данных**

Язык ZRV-2018 является строго типизированным, поэтому преобразование типов данных не поддерживается.

* 1. **Идентификаторы**

Язык ZRV-2018 поддерживает идентификаторы, содержащие только символы нижнего регистра латинского алфавита, цифры от 0 до 9, а также символ ‘\_’. Максимальная длина имени – 16 символов. Идентификаторы не должны совпадать с ключевыми словами языка ZRV-2018.

Примеры неправильного идентификатора: numb, My string, временное\_значение.

Примеры правильного идентификатора: numb1, my\_string, temp\_value.

* 1. **Литералы**

В языке ZRV-2018 имеется 2 типа литералов: целочисленные и строковые. Все литералы являются rvalue. Описание литералов представлено в таблице 1.3.

Таблица 1.3 – Литералы языка ZRV-2018

|  |  |
| --- | --- |
| Тип литерала | Пояснение |
| Целочисленные | Целочисленные неотрицательные литералы. Состоят из цифр от 0 до 9. По умолчанию инициализируются 0. |
| Строковые | Строка символов, заключённая в кавычки **" "**. Состоят из прописных и строковых символов русского и латинского алфавита, чисел от 0 до 9, символов: ‘ ’ (пробел), '.', ',', ‘?’, ‘!’, ‘; ’, ‘: ’, ‘-‘, ‘ (‘, ‘) ’. По умолчанию инициализируются пустой строкой . |

* 1. **Объявление данных**

Для объявления переменной используется ключевое слово new. После ключевого слово мы указываем тип данных и имя идентификатора.

Пример с типом numb: newnumber numb.

Пример с типом letter: new string letter.

Все переменные должны находиться внутри программного блока языка. Допускается объявление одинаковых переменных в разных блоках языка, то есть все переменные локальные. Каждая переменная или параметр функции дополнительно получает префикс в виде названия функции, в которой она объявлена.

* 1. **Инициализация данных**

Инициализация данных происходит посредством использования оператора присваивания: ‘=’. После оператора присваивания следует литерал или другой идентификатор соответствующего типа. При этом значение переменной можно менять в ходе работы программы. Так же есть значения по умолчанию при объявлении переменной без инициализации. Для целочисленного типа данных: значение 0. Для строкового типа данных: строка нулевой длины **" "** (пустая строка).

* 1. **Инструкции языка**

Инструкции языка ZRV-2018 представлены в таблице 1.4.

Таблица 1.4 – Инструкции языка ZRV-2018

|  |  |
| --- | --- |
| Инструкция | Запись на языке ZRV-2018 |
| Объявление переменной | new <идентификатор> <тип данных>; |
| Присваивание | <идентификатор> = <значение/идентификатор>; |
| Объявление внешней функции | function <идентификатор> (<идентификатор> <тип данных> , …) <тип данных> {…} |
| Блок инструкций | block {…} |
| Вывод данных | escape (<идентификатор/литерал>) |
| Условный оператор | if (<логическое выражение>) next {…} else {…} |
| Возврат из подпрограммы | return <идентификатор/литерал> |

* 1. **Операции языка**

Арифметические операции языка ZRV-2018 представлены в таблице 1.5.

Таблица 1.5 – Операции языка ZRV-2018

|  |  |
| --- | --- |
| Операция | Приоритет |
| С целочисленным типом данных | + – сложение, приоритет 3  - – вычитание, приоритет 3  \* – умножение, приоритет 2  = – присваивание |
| Со строковым типом данных | **= -** присваивание |
| (…) | Приоритет 0 или 4 |
| , | Приоритет 1 |

Наибольший приоритет равен нулю, наименьший равен 4

* 1. **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. Так же предусмотрено использование логических выражений в условном операторе if…next…else.

* 1. **Конструкция языка**

Конструкции языка ZRV-2018 представлены в таблице 1.6.

Таблица 1.6 – Конструкции языка ZRV-2018

|  |  |
| --- | --- |
| Название функции | Запись на языке ZRV-2018 |
| Главная функция (точка входа в приложение) | block {…} |
| Внешняя функция | function <идентификатор> (<идентификатор> <тип данных>, …) <тип данных>  {  …  return <выражение>;  } |

* 1. **Область видимости идентификаторов**

Переменные обязаны быть внутри программного блока функций. Глобальные переменные не предусмотрены.

* 1. **Сематические проверки**

Правила:

* Повторное объявление идентификатора не допускается;
* Идентификатор должен быть объявлен до его использования;
* Тип передаваемых в функцию стандартной библиотеки данных должен совпадать с типом её параметров;
* Операнды в арифметическом выражении не могут быть разных типов;
* Проверка на максимальное и минимальное значение целочисленного литерала. Проверка на максимальную длину строкового литерала.
  1. **Распределение оперативной памяти на этапе выполнения**

Все переменные размещаются в куче.

* 1. **Стандартная библиотека и её состав**

Состав функций, входящих в стандартную библиотеку языка ZRV-2018 представлен в таблице 1.7.

Таблица 1.7 – Состав стандартной библиотеки языка ZRV-2018

|  |  |  |  |
| --- | --- | --- | --- |
| Имя функции | Возвращаемое значение | Принимаемые параметры | Описание |
| length | numb | Строка типа letter | Функция вычисляет длину строки. |
| subline | numb | numb a - число  numb b - степень | Функция возводит а в степень b. |

* 1. **Ввод и вывод данных**

В языке ZRV-2018 не предусмотрен ввод данных. Вывод в стандартный поток вывода – escape **(**<идентификатор/литерал>**)**.

* 1. **Точка входа**

Точкой входа в языке ZRV-2018 является функция block.

* 1. **Препроцессор**

Препроцессор в языке ZRV-2018 отсутствует.

* 1. **Соглашения о вызовах**

В языке ZRV-2018 вызов функций происходит по соглашению о вызовах stdcall:

* все параметры функции передаются через стек;
* занесение в стек параметров идёт справа налево;
* память высвобождает вызываемый код.
  1. **Объектный код**

Язык ZRV-2018 транслируется в язык ассемблера, после чего в объектный код.

* 1. **Классификация сообщений транслятора**

Классификация сообщений транслятора языка приведена в таблице 1.8.

Таблица 1.8 – Классификация сообщений транслятора

|  |  |
| --- | --- |
| Интервал | Описание ошибок |
| От 0 до 9 | Системные ошибки |
| От 10 до 19 | Ошибки параметров |
| От 20 до 29 | Ошибки открытия и чтения файлов |
| От 30 до 49 | Ошибки лексического анализа |
| От 50 до 79 | Ошибки синтаксического анализа |
| От 80 до 99 | Ошибки семантического анализа |

* 1. **Контрольный пример**

Контрольный пример представлен в главе Приложения.

2 Структура транслятора



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

Транслятор преобразует программу, написанную на языке ZRV-2018 в программу на языке ассемблера. Компонентами транслятора являются: лексический анализатор, семантический анализатор, синтаксический анализатор, генератор кода на язык ассемблера. Принципы их взаимодействия представлены на рисунке 2.1.

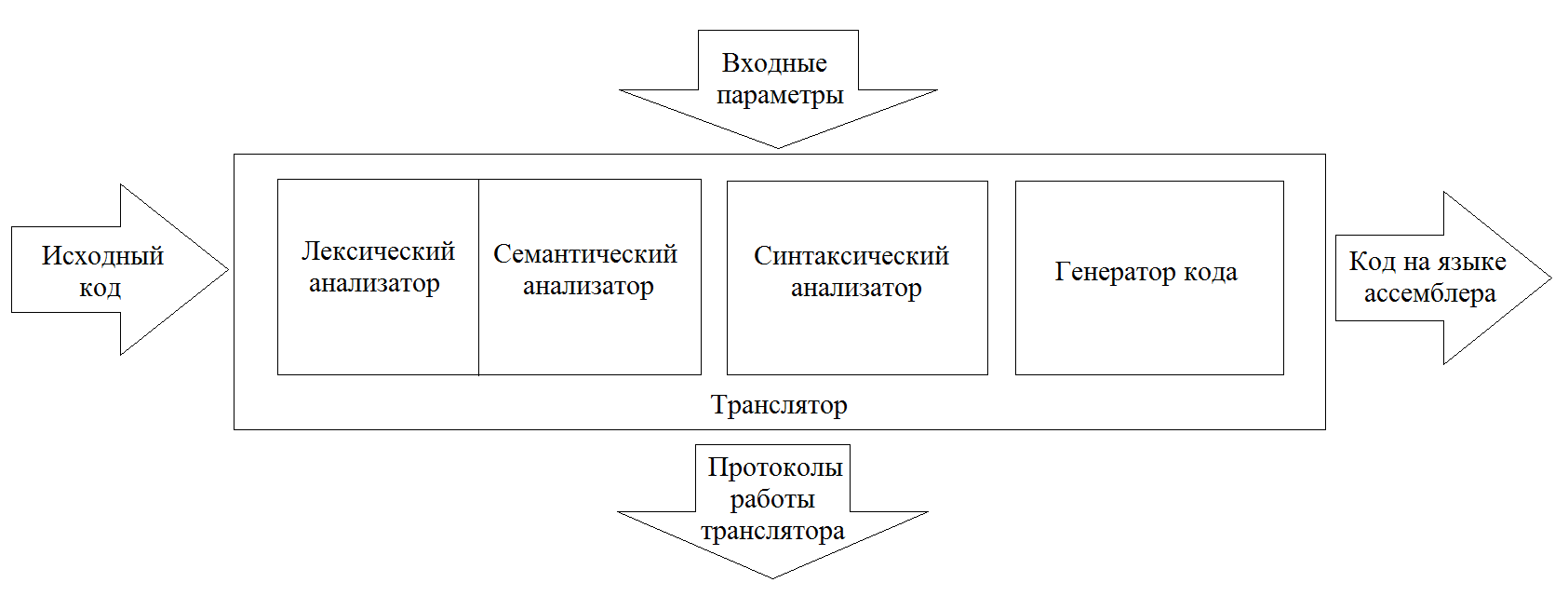


Рисунок 2.1 – Структура транслятора

Лексический анализатор читает исходную программу и выделяет в ее тексте лексемы входного языка, а также лексические ошибки. На вход лексического анализатора поступает текст исходной программы. Результатом работы лексического анализатора является перечень всех найденных в тексте исходной программы лексем из которых формируется таблица лексем.

Семантический анализатор проверяет код исходной программы на правильность оформления со стороны семантики, выделяет семантические отношения и формирует семантическое представления текста.

Лексический и семантический анализаторы в трансляторе ZRV-2018 реализованы как единое целое и выполняются одновременно.

Синтаксический анализатор распознаёт синтаксические конструкции, формирует промежуточный код, выявляет синтаксические ошибки и строит дерево разбора. Входным параметром синтаксического анализатора является таблица лексем.

Генератор кода выполняет трансляцию кода с языка ZRV-2018 в код на языке ассемблера из полученных на предыдущих стадиях обработки данных.

* 1. **Перечень входных параметров транслятора**

Входные параметры представлены в таблице 2.1.

Таблица 2.1 Входные параметры транслятора языка ZRV-2018

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<name> | Входной файл с расширением .txt, в котором содержится исходный код на языке ZRV-2018 | Не предусмотрено |
| -log:<name> | Протокол результата проверки входного файла на допустимость символов | log.log |
| -lex:<name> | Протокол работы лексического и семантического анализа. | lex.lex |
| -sin:<name> | Протокол работы синтаксического анализа. | sin.sin |
| -out:<name> | Файл для записи результата работы транслятора | out.asm |
| -trg | Ключ для вывода трассировки синтаксического анализа в протокол .sin | Ключа нет, т.е. трассировка не записывается |

* 1. **Перечень протоколов, формируемых транслятором и их содержимое**

Перечень протоколов, формируемых в результате работы транслятора языка ZRV-2018, а также их описание представлены в таблице 2.2.

Таблица 2.2 – Протоколы, формируемые транслятором языка ZRV-2018

|  |  |
| --- | --- |
| Формируемый протокол | Описание протокола |
| Файл журнала с параметром log | Содержит информацию о входных параметрах и о проверки символов на допустимость. |
| <name>.lex | Результат работы лексического и семантического анализа. Содержит таблицы лексем и идентификаторов. |
| <name>.sin | Результат работы синтаксического анализа. Содержит правила разбора, трассировку, а также преобразованные после польской записи таблицы лексем и идентификаторов. |
| Выходной файл c параметром out | Содержит сгенерированный код на языке Ассемблера. |

3 Разработка лексического анализатора

* 1. **Структура лексического анализатора**

Лексический анализатор – часть транслятора, выполняющая лексический анализ. Лексический анализатор принимает обработанный и разбитый на отдельные компоненты исходный код на языке ZRV-2018. На выходе формируется таблица лексем и таблица идентификаторов. Структура лексического анализатора представлена на рисунке 3.1

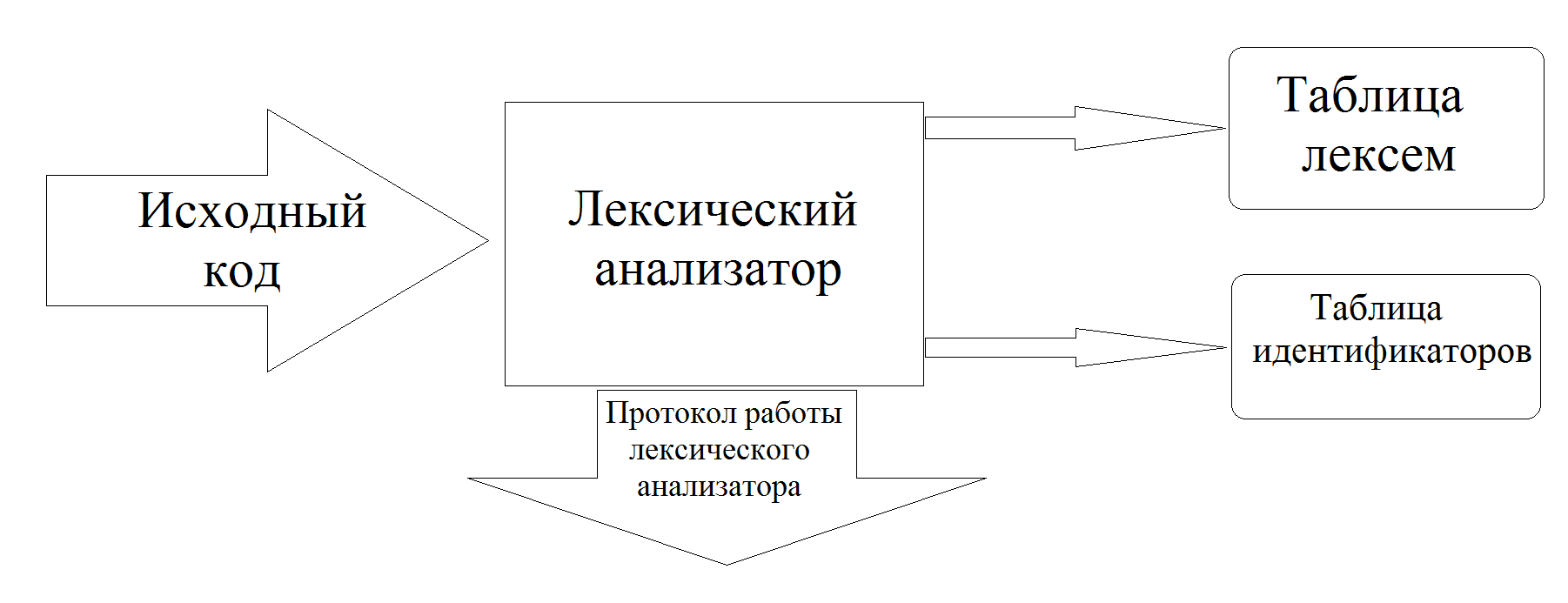
****

Рисунок 3.1 Структура лексического анализатора ZRV-2018

* 1. **Контроль входных символов**

Таблица входных символов представлена на рисунке 3.2.

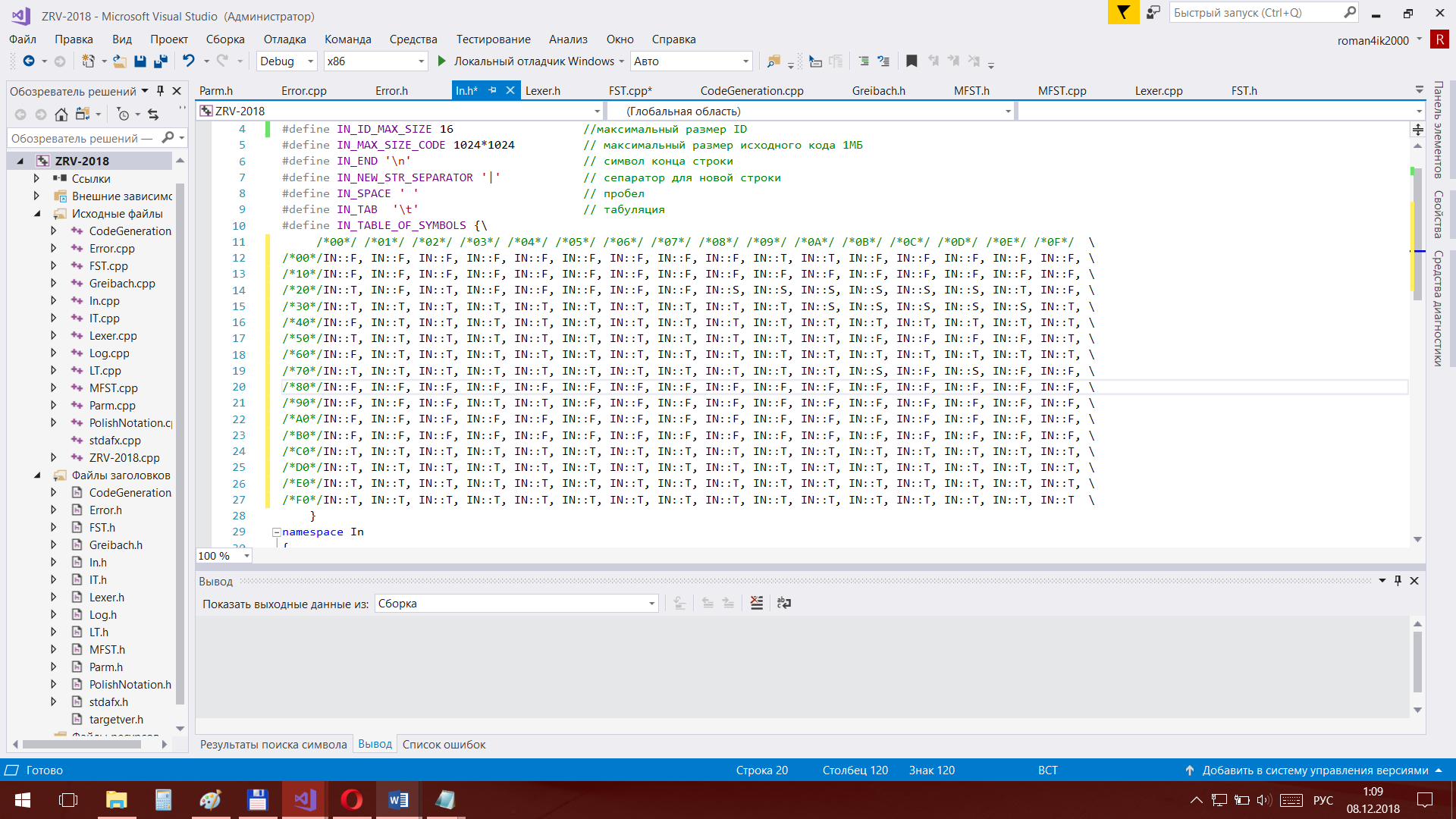


Рисунок 3.2 Таблица входных символов языка ZRV-2018

Каждый элемент таблицы на рисунке 3.2 соответствует символу в шестнадцатеричной системе счисления в таблице кодировки Windows-1251.

При этом T – разрешённый символ, F – запрещённый символ, S – символ-сепаратор.

* 1. **Удаление избыточных символов**

Избыточными символами в языке ZRV-2018 являются символы пробела и табуляции. Их удаление происходит посредством игнорирования. То есть посимвольно считываем файл с исходным кодом программы. При встрече символа пробела или табуляции они трактуются как символы-сепараторы, но данные символы в очередь лексем не записываются.

## **Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов**

Соответствие лексем ключевым словам, операндам и сепараторам представлено в таблице 3.1.

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

|  |  |  |
| --- | --- | --- |
| Тип цепочки | Цепочка | Лексема |
| Ключевые слова | numb | t |
| letter | t |
| new | w |
| function | f |
| block | b |
| return | r |
| if | ? |
| next | 1 |
| else | 2 |
| escape | e |
| length | s |
| subline | p |
| Сепараторы | ; | ; |
| , | , |
| { | { |
| } | } |
| ( | ( |
| ) | ) |
| Другие | Идентификатор | i |
| Целочисленный литерал | n |
| Строковый литерал | l |
| Оператор | o |
| Операторы | = | = |
| < | < |
| > | > |
| + | o |
| - | o |
| \* | o |

Реализация таблицы лексем, конечных автоматов, соответствующих лексемам языка ZRV-2018, представлена в приложении А.

* 1. **Основные структуры данных**

Основные структуры данных, а именно таблица лексем и таблица идентификаторов языка ZRV-2018 представлены в приложении А. Таблица лексем содержит лексему, номер лексемы, полученный при разборе, номер строки в исходном коде и приоритет. Таблица идентификаторов содержит имя идентификатора, номер в таблице лексем, тип данных, тип идентификатора и его значение.

* 1. **Принцип обработки ошибок**

Имеется 2 типа ошибок: предупреждения и критические ошибки.

При возникновении предупреждения транслятор продолжает свою работу, записывая предупреждения с номером ошибки и сообщением в отдельную структуру.

При возникновении критической ошибки или если количество предупреждений больше 8, то работа транслятора прекращается и в файл выводится соответствующее номеру ошибки сообщение.

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

Перечень сообщений лексического анализатора:

* Ошибка лексики: Цепочка символов не разобрана;
* Ошибка лексики: Таблица лексем переполнена;
* Ошибка лексики: Таблица идентификаторов переполнена;
* Ошибка лексики: Два арифметических знака подряд;

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

Входным параметром лексического анализа является очередь структур, заполненная полями, состоящими из лексем и соответствующих им номеров строк в исходном файле, которые получаются в процессе проверки исходного кода на допустимость символов и разбиения его на лексемы.

## **Алгоритм лексического анализа**

Лексический анализ выполняется лексическим анализатором. Лексический анализатор принимает обработанный и разбитый на отдельные компоненты исходный код на языке ZRV-2018. Он выделяет и классифицирует лексемы в тексте исходной программы. На выходе формируется таблица лексем и таблица идентификаторов. Алгоритм лексического анализатора основывается на работе конечных автоматов, который можно представить в виде графов. Регулярные выражения – это способ задания регулярных языков. Они состоят из символов, констант и операторов, которые определяют множество строк и множество операций над ними. Любое регулярное выражение можно представить в виде графа.

Пример. Регулярное выражение для ключевого слова numb.

Граф конечного автомата для цепочки numb представлен на рисунке 3.3, где S0 – начальное состояние, S4 – конечное состояние автомата.



Рисунок 3.3 – Граф переходов для цепочки numb

## **Контрольный пример**

Результаты работы лексического анализатора (таблицы лексем и таблицы идентификаторов) представлены в приложении А.

4 Разработка синтаксического анализатора



## **Структура синтаксического анализатора**

Синтаксический анализатор – часть транслятора, выполняющая синтаксический анализ. Выполняется синтаксический анализ после лексического анализа. Синтаксический анализатор распознаёт синтаксические конструкции, формирует промежуточный код, выявляет синтаксические ошибки и строит дерево разбора. Входным параметром синтаксического анализатора является таблица лексем и таблица идентификаторов, полученные после лексического анализа. Его структура представлена на рисунке 4.1.



Рисунок 4.1 – Структура синтаксического анализатора

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

Синтаксический анализатор транслятора языка ZRV-2018 использует контекстно-свободную грамматику , где

T – множество терминальных символов (приведены в таблице 3.1),

N – множество нетерминальных символов,

P – множество правил языка, где левая часть состоит из одного нетерминала,

S – стартовый символ грамматики, являющийся нетерминалом.

Эта грамматика имеет нормальную форму Грейбах, т.к. она не леворекурсивная (не содержит леворекурсивных правил, т.е. содержащих вызов самих себя без продвижения по строке) 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 ZRV-2018 представлена в приложении Б.

TS – терминальные символы, а именно некоторые знаки операций, символы-сепараторы, некоторые строчные символы латинского алфавита.

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

Ниже приведена таблица 4.1, содержащая нетерминалы с соответствующими им правилами.

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

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| A | fi(F)t{N}A  b{N} | Порождает правила, описывающее общую структуру программы |
| N | wF;  wF;N  rR;  i=O;  i=O;N  e(E);  e(E);N  ?Q1{N}2{N}N  ?Q1{N}N | Порождает правила, описывающие инструкции языка |
| F | it  it,F  i,F | Порождает правила, описывающие параметры локальной функции при её объявлении и само объявление |
| O | i  l  n  iM  nM  p(P)  p(P)M  s(S)  s(S)M  i(I)  i(I)M  (O)  (O)M | Порождает правила, описывающие выражения |
| E | i  l  n | Порождает правила, описывающие параметры стандартной функции escape |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| I | i,I  i  n  n,I  l  l,I | Порождает правила, описывающие параметры вызываемой функции |
| P | i,n  n,n | Порождает правила, описывающие параметры стандартной функции substring |
| S | i  l | Порождает правила, описывающие параметры стандартной функции lenght |
| M | oO  oOM | Порождает правила, описывающие арифметические операторы |
| Q | (R<R)  (R>R) | Порождает правила, описывающие операнды оператора условия |
| R | i  n | Порождает правила, описывающие параметр выражения return |

## **Построение конечного магазинного автомата**

Конечный автомат с магазинной памятью является набором , где:

*  - конечное множество состояний автомата. Состояние автомата – это структура, которая содержит номер текущего правила и цепочки, стек автоматов, позицию на входной ленте;
*  - допустимый входной алфавит. Алфавит – это набор терминальных и нетерминальных символов;
*  - алфавит памяти (магазина). Содержит маркер дна стека и стартовый символ;
*  - функция переходов автомата. Она представляет из себя набор правил грамматики, описанных в таблице 4.1;
*  - начальное состояние автомата. Единственное допустимое начальное состояние. Его принимает автомат в начале своей работы. В языке ZRV-2018 является нетерминальным символом A;
*  - конечное состояние (дно стека). В языке ZRV-2018 символ маркера дна стека ‘|’;
*  - множество конечных состояний. Конечным состоянием является пустой магазин автомата, а также совпадение позиции на входной ленте с размером ленты.

## **Основные структуры данных**

Основные структуры данных синтаксического анализатора включают в себя структуру грамматики Грейбах и структуру конечного автомата с магазинной памятью. Данные структуры представлены в приложении В.

## **Описание алгоритма синтаксического разбора**

Синтаксический разбор выполняется синтаксическим анализатором. Синтаксический анализатор принимает таблицу лексем и таблицу идентификаторов. Он распознаёт синтаксические конструкции и формирует промежуточный код. На выходе формируется дерево разбора. Алгоритм синтаксического анализатора основывается на работе конечных автоматов с магазинной памятью который выглядит следующим образом:

1. В магазин автомата записывается стартовый символ.
2. Формируется входная лента из полученных ранее таблиц лексем и идентификаторов.
3. Автомат начинает работу.
4. Выбирается цепочка, соответствующая нетерминалу, записывается в обратном порядке в магазин.
5. Если терминал и в стеке, и в ленте совпадает, то этот терминал удаляется из стека и ленты. В противном случае возвращаемся в предыдущее состояние и выбираем другую цепочку нетерминала.
6. Если в магазине встретился нетерминал, то снова выбирается цепочка, соответствующая данному нетерминалу (см. пункт 4).
7. Если символ достиг дна стека в момент, когда лента оказалась пуста, то выполнение синтаксического анализа успешно завершено, иначе генерируется исключение.

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

Перечень сообщений синтаксического анализатора:

* Ошибка синтаксиса: Неверная структура программы;
* Ошибка синтаксиса: Неверный оператор;
* Ошибка синтаксиса: Неверное выражение;
* Ошибка синтаксиса: Ошибка в операторе объявления или в параметрах функции;
* Ошибка синтаксиса: Ошибка в параметре вызываемой функции escape стандартной библиотеки;
* Ошибка синтаксиса: Ошибка в параметре вызываемой функции length стандартной библиотеки;
* Ошибка синтаксиса: Ошибка в параметрах вызываемой функции subline стандартной библиотеки;
* Ошибка синтаксиса: Ошибка в параметрах вызываемой функции;
* Ошибка синтаксиса: Ошибка арифметического оператора;
* Ошибка синтаксиса: Ошибка в операторе условия;
* Ошибка синтаксиса: Ошибка в возвращаемом выражении;
* Ошибка синтаксиса: Отсутствует конец правила;
* Ошибка синтаксиса: Цепочка разобрана не полностью (стек не был освобождён).
  1. Параметры синтаксического анализатора и режимы его работы

Выходным параметром синтаксического анализатора является таблица лексем и таблица идентификаторов, сформированных на этапе лексического анализа. А также для работы синтаксического анализатора применяются правила контекстно-свободной грамматики в форме Грейбах.

Выходными параметрами являются дерево разбора (трассировка) и правила разбора, которые записываются в протокол синтаксического анализатора.

* 1. Принцип обработки ошибок

Алгоритм обработки ошибок на фазе синтаксического анализа работает следующем образом: если невозможно сопоставить входной конструкции подходящую цепочку, то генерируется соответственная ошибка. Все ошибки записываются в общую структуру ошибок и по окончанию трассировки будут выведены в протокол.

* 1. Контрольный пример

Результат работы синтаксического анализа, а также дерево разбора представлены в приложении Г.

5 Разработка семантического анализатора

* 1. Структура семантического анализатора

Семантический анализатор входит в состав лексического анализатора и выполняется вместе с ним. Реализуется он в виде отдельных проверок на нахождение в особом месте программы с помощью флагов.

* 1. Функции семантического анализатора

Семантический анализатор выделяет семантические отношения, формирует семантическое представление текста, производит проверку на основные правила языка (см. пункт 1.16).

* 1. Структура и перечень сообщений семантического анализатора

Перечень сообщений семантического анализатора:

* Ошибка семантики: Повторное объявление идентификатора;
* Ошибка семантики: Ошибка в типе идентификатора;
* Ошибка семантики: Ошибка в передаваемых в функцию значениях;
* Ошибка семантики: Ошибка в параметре вызываемой функции length стандартной библиотеки;
* Ошибка семантики: Необъявленный идентификатор;
* Ошибка семантики: Несоответствие типов в операторе присваивания;
* Ошибка семантики: Превышение предельно допустимого значения numb литерала;
  1. Принцип обработки ошибок

Имеется 2 типа ошибок: предупреждения и критические ошибки.

При возникновении предупреждения транслятор продолжает свою работу, записывая предупреждения с номером ошибки и сообщением в отдельную структуру.

При возникновении критической ошибки или если количество предупреждений больше 8, то работа транслятора прекращается и в файл выводится соответствующее номеру ошибки сообщение.

* 1. Контрольный пример

Так как семантический анализатор входит в состав лексического анализатора, и таблицы идентификаторов и лексем проходят через них одновременно, то результат работы представлен в приложении А, где показан результат лексического анализатора.

6 Преобразование выражений

* 1. Выражения, допустимые языком

В языке ZRV-2018 допускаются выражения, применимые к целочисленному типу данных. В выражениях поддерживаются арифметические операции, такие как сложение (+), вычитание (-), умножение (\*) и (), и вызовы функций как операнды арифметических выражений.

Приоритет операций представлен в таблице 6.1.

Таблица 6.1 – Приоритет операций в языке ZRV-2018

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

* 1. Польская запись

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

Польская запись – это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок, что облегчает работу с выражениями.

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

Алгоритм построения:

* исходная строка просматривается слева направо;
* операнды переносятся в результирующую строку в порядке их следования;
* операция записывается в стек, если стек пуст или в вершине стека лежит отрывающая скобка;
* операция выталкивает все операции с большим или равным приоритетом в результирующую строку;
* запятая не помещается в стек, если в стеке операции, то все выбираются в строку;
* отрывающая скобка помещается в стек;
* закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются;
* закрывающая скобка с приоритетом, равным 4, выталкивает все до открывающей с таким же приоритетом и генерирует # – специальный символ, в который записывается информация о вызываемой функции, а в поле приоритета для данной лексемы записывается число параметров вызываемой функции;
* по концу разбора исходной строки все операции, оставшиеся в стеке, выталкиваются в результирующую строку.

Пример преобразования представлен в таблице 6.2.

Таблица 6.2 – Пример преобразования выражения в обратную польскую запись

|  |  |  |
| --- | --- | --- |
| Исходная строка | Результирующая строка | Стек |
| n\*8 + f(i) |  |  |
| \*2 + f(i) | n |  |
| 2 + f(i) | n | \* |
| + f(i) | n8 | \* |
| f(i) | n8\* | + |
| (i) | n8\* | + |
| i) | n8\* | +( |
| ) | n8\*i | + |
|  | n8\*i#1+ |  |

* 1. Программная реализация обработки выражений

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

* 1. Контрольный пример

Контрольный пример преобразования выражения на основе имеющихся таблиц лексем и идентификаторов с обратную польскую запись представлен в приложении Д.

7 Генерация кода

* 1. Структура генератора кода

Генератор кода выполняет трансляцию кода, т.е. переводит внутреннее представление исходной программы в цепочку символов языка ассемблера. На вход генератора поступают таблицы лексем и идентификаторов на основе которых генерируется файл с ассемблерным кодом.



Рисунок 7.1 Структура генератора кода

* 1. Представление типов данных в оперативной памяти

Элементы таблицы идентификаторов расположены в сегментах данных и сегментах констант языка ассемблера (.data и .const.). Идентификаторы языка ZRV-2018 размещены в сегменте данных(.data). Литералы – в сегменте констант (.const). Соответствия между типами данных идентификаторов на языке ZRV-2018 и на языке ассемблера приведены в таблице 7.1.

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

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке ZRV-2018 | Тип идентификатора на языке ассемблера | Пояснение |
| numb | SDWORD | Хранит целочисленный тип данных со знаком. |
| letter | DWORD | Хранит указатель на начало строки. |
| №(номер) | BYTE  DWORD | Символьные и целочисленные литералы |

* 1. Статическая библиотека

Состав статической библиотеки приведён в таблице 7.2.

Таблица 7.2 – Состав статической библиотеки

|  |  |
| --- | --- |
| Функция | Назначение |
| length(letter) | Находит длину строки типа letter |
| subline(numb, n) | Возводит число типа numb в указанную степень n |
| escape(…) | Стандартный вывод в поток.  Отдельно реализована для каждого типа.  escape\_n(numb) – вывод целочисленных значений в поток.  escape\_l(letter) – вывод символьных значений в поток. |

* 1. Алгоритм работы генератора кода

Через генератор кода проходят таблица лексем и таблица идентификаторов, встретив определённую лексему и определив, в каком месте кода она находится, генератор стразу же транслирует эту лексему на язык Ассемблера. Генерируемый код записывается в файл объектного кода (рисунок 7.1). Пример алгоритма, а также контрольный пример представлен в приложении Е.

8 Тестирование транслятора

* 1. Тестирование фазы проверки на допустимость символов

В языке ZRV-2018 не разрешается использовать запрещённые символы (рисунок 3.2)

Результат использования такого символа представлен в таблице 8.1.

Таблица 8.1 – Тестирование фазы проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| block {  new c/ numb;  } | Ошибка 23: Ошибка проверки входного файла: Недопустимый символ в исходном файле, строка 2, позиция 5 |

* 1. Тестирование лексического анализатора

В языке ZRV-2018 на этапе лексического анализа могут возникать ошибки, которые описаны в пункте 3.7. Результат тестирования представлен в таблице 8.2.

Таблица 8.2 – Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| block {  new c numb;  c = 2 +- 3;  } | Ошибка 33: Ошибка лексики: Два арифметических знака подряд, строка 3, позиция -1 |

* 1. Тестирование синтаксического анализатора

В языке ZRV-2018 на этапе синтаксического анализа могут возникать 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 |
| block {  new a numb;  a = length ();  } | Ошибка 56: Ошибка синтаксиса: Ошибка в параметре вызываемой функции length стандартной библиотеки, строка 3, позиция -1 |

* 1. Тестирование семантического анализатора

В языке ZRV-2018 на этапе семантического анализа могут возникать ошибки, описанные в пункте 5.3. Результаты тестирования семантического анализатора показаны в таблице 8.4.

Таблица 8.4 –Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| block {  new a numb;  new a numb;  a = 3;  } | Ошибка 80: Ошибка семантики: Повторное объявление идентификатора, строка 3, позиция -1 |

**Заключение**

**Список используемой литературы**

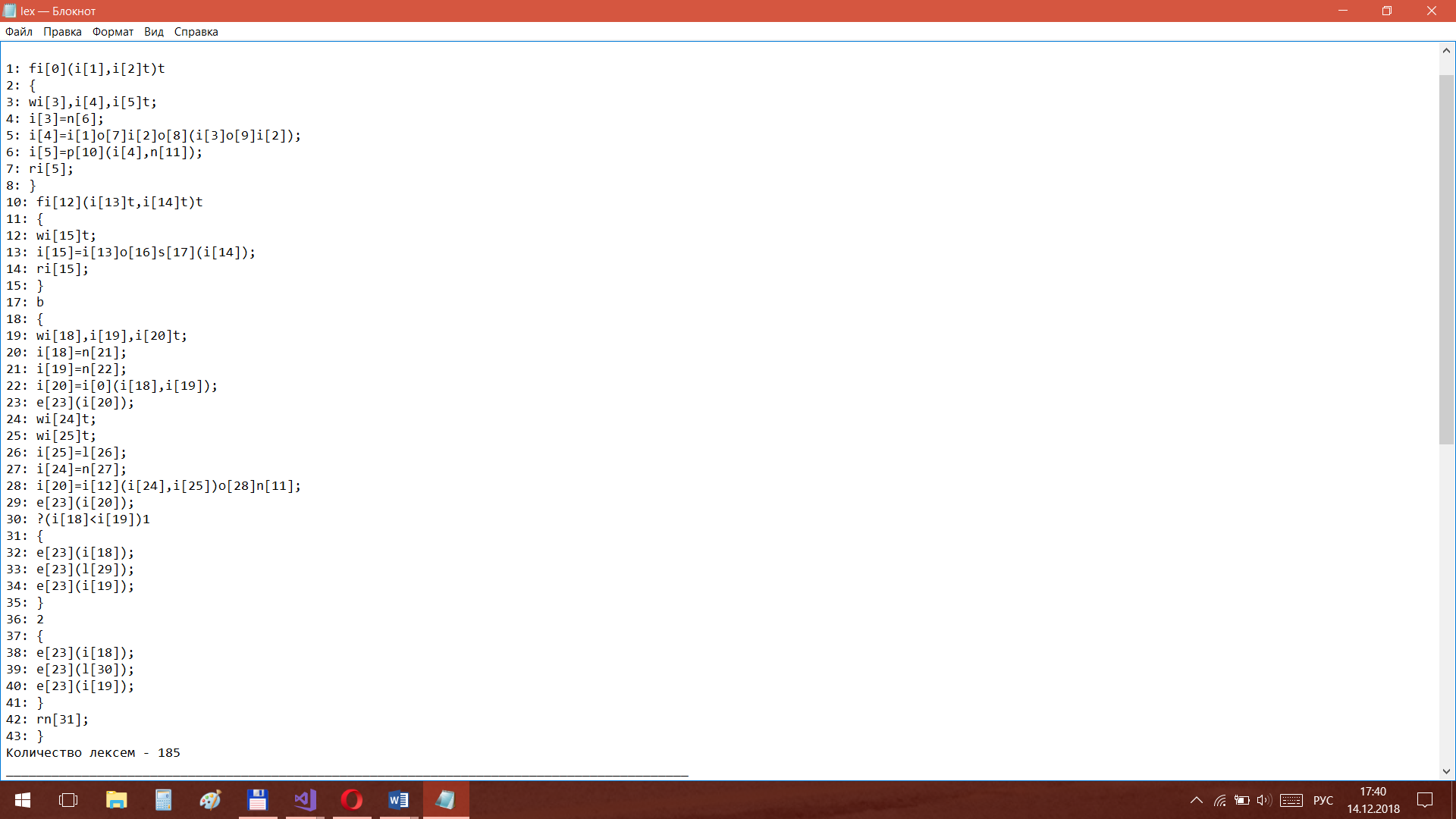
1. Серебряков В. А. Основы конструирования компиляторов / В. А. Серебряков, М. П. Галочкин. – М.: Эдиториал УРСС, 2001. – 192с.
2. Молчанов А. Ю. Системное программное обеспечение: Учебник для вузов / А. Ю. Молчанов. – СПб.: Питер, 2006. – 396с.
3. Документация Visual C++ [Электронный ресурс] / Microsoft, Corp. <https://docs.microsoft.com/ru-ru/cpp/standard-library/regular-expressions-cpp?view=vs-2017> – Дата доступа: 29.11.2018.
4. Ахо А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003. – 768с.

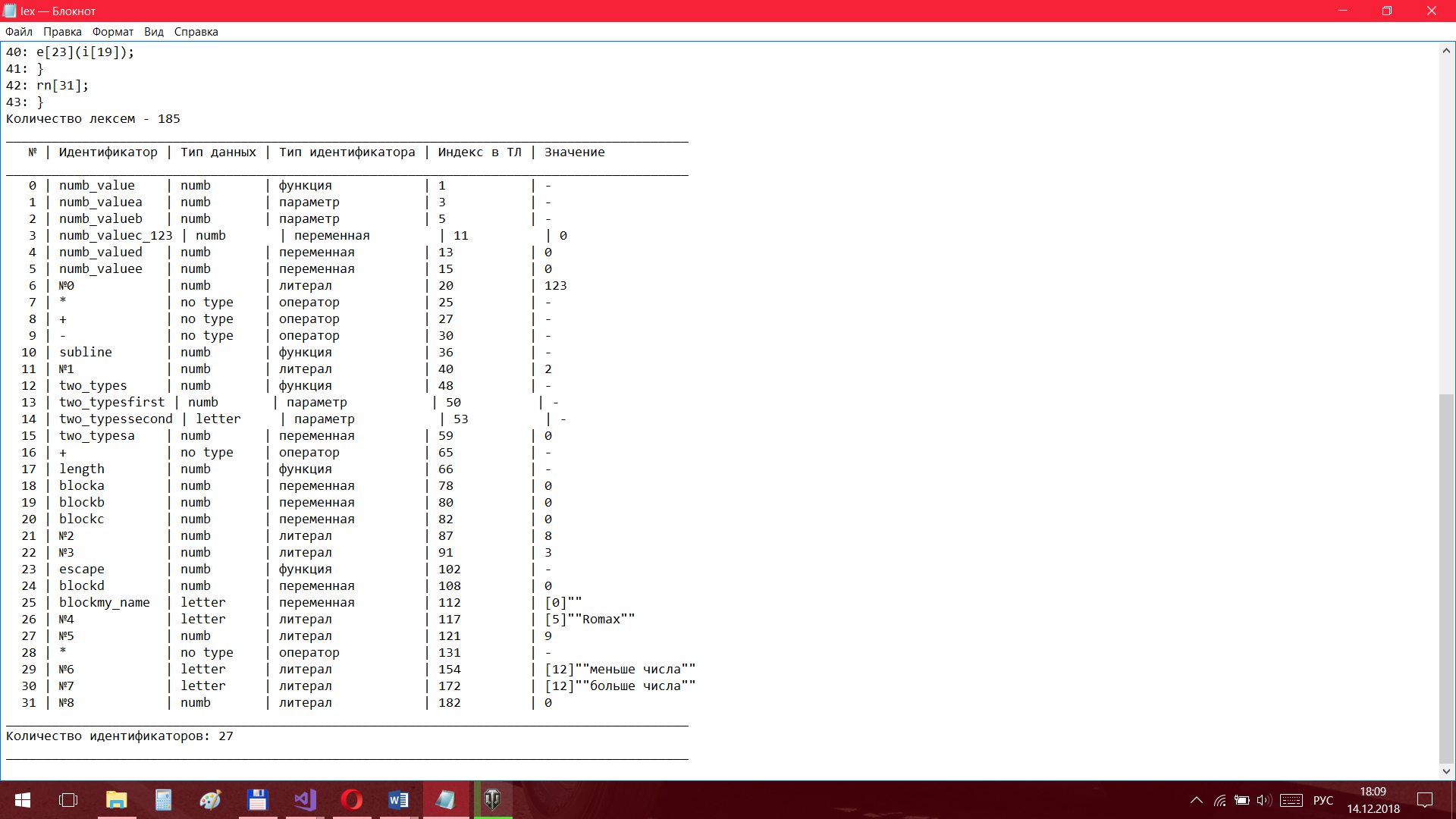
**Приложение А**

Контрольный пример

|  |
| --- |
| function numb\_value ( a , b numb) numb  {  new c\_123 , d , e numb ;  c\_123 = 123;  d = a \* b + ( c\_123 - b );  e = subline(d, 2);  return e;  }  function two\_types (first numb ,second letter ) numb  {  new a numb;  a = first + length ( second );  return a;  }  block  {  new a, b, c numb ;  a = 8;  b = 3;  c = numb\_value(a,b);  escape(c);  new d numb;  new my\_name letter;  my\_name = "Romax";  d = 9;  c = two\_types (d, my\_name) \* 2;  escape(c);  if (a<b) next  {  escape(a);  escape("меньше числа");  escape(b);  }  else  {  escape(a);  escape("больше числа");  escape(b);  }  return 0;  } |

**Приложение А**





|  |
| --- |
| FST\* arr\_of\_fst\_symbols()  {  FST fstOperator(LEX\_OPERATOR, SYMB\_OPERATORS, "", 2, NODE(3, RELATION('+', 1), RELATION('-', 1), RELATION('\*', 1)), \  NODE()  );  FST fstLeftHesis(LEX\_LEFTHESIS, SYMB\_FOR\_PRIORITY, "", 2, NODE(1, RELATION('(', 1)), \  NODE()  );  FST fstRightHesis(LEX\_RIGHTHESIS, SYMB\_FOR\_PRIORITY, "", 2, NODE(1, RELATION(')', 1)), \  NODE()  );  FST fstEqual(LEX\_EQUAL, SYMB\_LEXEMS, "", 2, NODE(1, RELATION('=', 1)), \  NODE()  );  FST fstLeftBrace(LEX\_LEFTBRACE, SYMB\_LEXEMS, "", 2, NODE(1, RELATION('{', 1)), \  NODE()  );  FST fstRightBrace(LEX\_RIGHTBRACE, SYMB\_LEXEMS, "", 2, NODE(1, RELATION('}', 1)), \  NODE()  );  FST fstSemicolon(LEX\_SEMICOLON, SYMB\_LEXEMS, "", 2, NODE(1, RELATION(';', 1)), \  NODE()  );  FST fstComma(LEX\_COMMA, SYMB\_FOR\_PRIORITY, "", 2, NODE(1, RELATION(',', 1)), \  NODE()  );  FST fstLess(LEX\_LESS, SYMB\_LEXEMS, "", 2, NODE(1, RELATION('<', 1)), \  NODE()  );  FST fstMore(LEX\_MORE, SYMB\_LEXEMS, "", 2, NODE(1, RELATION('>', 1)), \  NODE()  );  FST fstLitNumb(LEX\_NUMB, SYMB\_LITERAL, "", 2, NODE(10,  RELATION('0', 1), RELATION('1', 1), RELATION('2', 1),  RELATION('3', 1), RELATION('4', 1), RELATION('5', 1),  RELATION('6', 1), RELATION('7', 1), RELATION('8', 1),  RELATION('9', 1)),  NODE()  );  FST fstId(LEX\_ID, SYMB\_TABLE\_ID, "", 2, NODE(37, RELATION('a', 1), RELATION('b', 1), RELATION('c', 1), RELATION('d', 1),  RELATION('e', 1), RELATION('f', 1), RELATION('g', 1), RELATION('h', 1),  RELATION('i', 1), RELATION('j', 1), RELATION('k', 1), RELATION('l', 1),  RELATION('m', 1), RELATION('n', 1), RELATION('o', 1), RELATION('p', 1),  RELATION('q', 1), RELATION('r', 1), RELATION('s', 1), RELATION('t', 1),  RELATION('u', 1), RELATION('v', 1), RELATION('w', 1), RELATION('x', 1),  RELATION('y', 1), RELATION('z', 1), RELATION('0', 1), RELATION('1', 1),  RELATION('2', 1), RELATION('3', 1), RELATION('4', 1), RELATION('5', 1),  RELATION('6', 1), RELATION('7', 1), RELATION('8', 1), RELATION('9', 1), RELATION('\_', 1)),  NODE()  );  } FST\* arr\_of\_fst\_words()  {  FST fstNumb(LEX\_NUMB, SYMB\_LEXEMS, "", 5, NODE(1, RELATION('n', 1)),  NODE(1, RELATION('u', 2)),  NODE(1, RELATION('m', 3)),  NODE(1, RELATION('b', 4)),  NODE()  );  FST fstLetter(LEX\_LETTER, SYMB\_LEXEMS, "", 7, NODE(1, RELATION('l', 1)),  NODE(1, RELATION('e', 2)),  NODE(1, RELATION('t', 3)),  NODE(1, RELATION('t', 4)),  NODE(1, RELATION('e', 5)),  NODE(1, RELATION('r', 6)),  NODE()  );  FST fstFunction(LEX\_FUNCTION, SYMB\_LEXEMS, "", 9, NODE(1, RELATION('f', 1)),  NODE(1, RELATION('u', 2)),  NODE(1, RELATION('n', 3)),  NODE(1, RELATION('c', 4)),  NODE(1, RELATION('t', 5)),  NODE(1, RELATION('i', 6)),  NODE(1, RELATION('o', 7)),  NODE(1, RELATION('n', 8)),  NODE()  );  FST fstNew(LEX\_NEW, SYMB\_LEXEMS, "", 4, NODE(1, RELATION('n', 1)),  NODE(1, RELATION('e', 2)),  NODE(1, RELATION('w', 3)),  NODE()  );  FST fstBlock(LEX\_BLOCK, SYMB\_LEXEMS, "", 6, NODE(1, RELATION('b', 1)),  NODE(1, RELATION('l', 2)),  NODE(1, RELATION('o', 3)),  NODE(1, RELATION('c', 4)),  NODE(1, RELATION('k', 5)),  NODE()  );  FST fstReturn(LEX\_RETURN, SYMB\_LEXEMS, "", 7, NODE(1, RELATION('r', 1)),  NODE(1, RELATION('e', 2)),  NODE(1, RELATION('t', 3)),  NODE(1, RELATION('u', 4)),  NODE(1, RELATION('r', 5)),  NODE(1, RELATION('n', 6)),  NODE()  );  FST fstNumbLiteral(LEX\_NUMB, SYMB\_LITERAL, "", 2, NODE(20, RELATION('0', 0), RELATION('1', 0), RELATION('2', 0),  RELATION('3', 0), RELATION('4', 0), RELATION('5', 0),  RELATION('6', 0), RELATION('7', 0), RELATION('8', 0),  RELATION('9', 0),  RELATION('0', 1), RELATION('1', 1), RELATION('2', 1),  RELATION('3', 1), RELATION('4', 1), RELATION('5', 1),  RELATION('6', 1), RELATION('7', 1), RELATION('8', 1),  RELATION('9', 1)),  NODE()  );  FST fstId(LEX\_ID, SYMB\_TABLE\_ID, "", 2, NODE(74, RELATION('a', 0), RELATION('b', 0), RELATION('c', 0), RELATION('d', 0),  RELATION('e', 0), RELATION('f', 0), RELATION('g', 0), RELATION('h', 0),  RELATION('i', 0), RELATION('j', 0), RELATION('k', 0), RELATION('l', 0),  RELATION('m', 0), RELATION('n', 0), RELATION('o', 0), RELATION('p', 0),  RELATION('q', 0), RELATION('r', 0), RELATION('s', 0), RELATION('t', 0),  RELATION('u', 0), RELATION('v', 0), RELATION('w', 0), RELATION('x', 0),  RELATION('y', 0), RELATION('z', 0), RELATION('0', 0), RELATION('1', 0),  RELATION('2', 0), RELATION('3', 0), RELATION('4', 0), RELATION('5', 0),  RELATION('6', 0), RELATION('7', 0), RELATION('8', 0), RELATION('9', 0), RELATION('\_', 0),  RELATION('a', 1), RELATION('b', 1), RELATION('c', 1), RELATION('d', 1),  RELATION('e', 1), RELATION('f', 1), RELATION('g', 1), RELATION('h', 1),  RELATION('i', 1), RELATION('j', 1), RELATION('k', 1), RELATION('l', 1),  RELATION('m', 1), RELATION('n', 1), RELATION('o', 1), RELATION('p', 1),  RELATION('q', 1), RELATION('r', 1), RELATION('s', 1), RELATION('t', 1),  RELATION('u', 1), RELATION('v', 1), RELATION('w', 1), RELATION('x', 1),  RELATION('y', 1), RELATION('z', 1), RELATION('0', 1), RELATION('1', 1),  RELATION('2', 1), RELATION('3', 1), RELATION('4', 1), RELATION('5', 1),  RELATION('6', 1), RELATION('7', 1), RELATION('8', 1), RELATION('9', 1), RELATION('\_', 1)),  NODE()  );  FST fstLetterLiteral(LEX\_LETTER, SYMB\_LITERAL, "", 4, NODE(2, RELATION('"', 1), RELATION('"', 2)), \  NODE(276, \  RELATION('A', 1), RELATION('B', 1), RELATION('C', 1), RELATION('D', 1), RELATION('E', 1), RELATION('F', 1), \  RELATION('G', 1), RELATION('H', 1), RELATION('I', 1), RELATION('J', 1), RELATION('K', 1), RELATION('L', 1), \  RELATION('M', 1), RELATION('N', 1), RELATION('O', 1), RELATION('P', 1), RELATION('Q', 1), RELATION('R', 1), \  RELATION('S', 1), RELATION('T', 1), RELATION('U', 1), RELATION('V', 1), RELATION('W', 1), RELATION('X', 1), \  RELATION('Y', 1), RELATION('Z', 1), \  RELATION('a', 1), RELATION('b', 1), RELATION('c', 1), RELATION('d', 1), RELATION('e', 1), RELATION('f', 1), \  RELATION('g', 1), RELATION('h', 1), RELATION('i', 1), RELATION('j', 1), RELATION('k', 1), RELATION('l', 1), \  RELATION('m', 1), RELATION('n', 1), RELATION('o', 1), RELATION('p', 1), RELATION('q', 1), RELATION('r', 1), \  RELATION('s', 1), RELATION('t', 1), RELATION('u', 1), RELATION('v', 1), RELATION('w', 1), RELATION('x', 1), \  RELATION('y', 1), RELATION('z', 1), RELATION('1', 1), RELATION('2', 1), RELATION('3', 1), RELATION('4', 1), \  RELATION('5', 1), RELATION('6', 1), RELATION('7', 1), RELATION('8', 1), RELATION('9', 1), RELATION('0', 1), \  \  RELATION('А', 1), RELATION('Б', 1), RELATION('В', 1), RELATION('Г', 1), RELATION('Д', 1), RELATION('Е', 1), \  RELATION('Ё', 1), RELATION('Ж', 1), RELATION('З', 1), RELATION('И', 1), RELATION('Й', 1), RELATION('К', 1), \  RELATION('Л', 1), RELATION('М', 1), RELATION('Н', 1), RELATION('О', 1), RELATION('П', 1), RELATION('Р', 1), \  RELATION('С', 1), RELATION('Т', 1), RELATION('У', 1), RELATION('Ф', 1), RELATION('Х', 1), RELATION('Ц', 1), \  RELATION('Ч', 1), RELATION('Ш', 1), RELATION('Щ', 1), RELATION('Ъ', 1), RELATION('Ы', 1), RELATION('Ь', 1), \  RELATION('Э', 1), RELATION('Ю', 1), RELATION('Я', 1), \  RELATION('а', 1), RELATION('б', 1), RELATION('в', 1), RELATION('г', 1), RELATION('д', 1), RELATION('е', 1), \  RELATION('ё', 1), RELATION('ж', 1), RELATION('з', 1), RELATION('и', 1), RELATION('й', 1), RELATION('к', 1), \  RELATION('л', 1), RELATION('м', 1), RELATION('н', 1), RELATION('о', 1), RELATION('п', 1), RELATION('р', 1), \  RELATION('с', 1), RELATION('т', 1), RELATION('у', 1), RELATION('ф', 1), RELATION('х', 1), RELATION('ц', 1), \  RELATION('ч', 1), RELATION('ш', 1), RELATION('щ', 1), RELATION('ъ', 1), RELATION('ы', 1), RELATION('ь', 1), \  RELATION('э', 1), RELATION('ю', 1), RELATION('я', 1), RELATION(' ', 1), RELATION('.', 1), RELATION(',', 1), \  RELATION('?', 1), RELATION('!', 1), RELATION(';', 1), RELATION(':', 1), RELATION('-', 1), RELATION(')', 1), \  RELATION('(', 1), \  \  RELATION('A', 2), RELATION('B', 2), RELATION('C', 2), RELATION('D', 2), RELATION('E', 2), RELATION('F', 2), \  RELATION('G', 2), RELATION('H', 2), RELATION('I', 2), RELATION('J', 2), RELATION('K', 2), RELATION('L', 2), \  RELATION('M', 2), RELATION('N', 2), RELATION('O', 2), RELATION('P', 2), RELATION('Q', 2), RELATION('R', 2), \  RELATION('S', 2), RELATION('T', 2), RELATION('U', 2), RELATION('V', 2), RELATION('W', 2), RELATION('X', 2), \  RELATION('Y', 2), RELATION('Z', 2), \  RELATION('a', 2), RELATION('b', 2), RELATION('c', 2), RELATION('d', 2), RELATION('e', 2), RELATION('f', 2), \  RELATION('g', 2), RELATION('h', 2), RELATION('i', 2), RELATION('j', 2), RELATION('k', 2), RELATION('l', 2), \  RELATION('m', 2), RELATION('n', 2), RELATION('o', 2), RELATION('p', 2), RELATION('q', 2), RELATION('r', 2), \  RELATION('s', 2), RELATION('t', 2), RELATION('u', 2), RELATION('v', 2), RELATION('w', 2), RELATION('x', 2), \  RELATION('y', 2), RELATION('z', 2), RELATION('1', 2), RELATION('2', 2), RELATION('3', 2), RELATION('4', 2), \  RELATION('5', 2), RELATION('6', 2), RELATION('7', 2), RELATION('8', 2), RELATION('9', 2), RELATION('0', 2), \  \  RELATION('А', 2), RELATION('Б', 2), RELATION('В', 2), RELATION('Г', 2), RELATION('Д', 2), RELATION('Е', 2), \  RELATION('Ё', 2), RELATION('Ж', 2), RELATION('З', 2), RELATION('И', 2), RELATION('Й', 2), RELATION('К', 2), \  RELATION('Л', 2), RELATION('М', 2), RELATION('Н', 2), RELATION('О', 2), RELATION('П', 2), RELATION('Р', 2), \  RELATION('С', 2), RELATION('Т', 2), RELATION('У', 2), RELATION('Ф', 2), RELATION('Х', 2), RELATION('Ц', 2), \  RELATION('Ч', 2), RELATION('Ш', 2), RELATION('Щ', 2), RELATION('Ъ', 2), RELATION('Ы', 2), RELATION('Ь', 2), \  RELATION('Э', 2), RELATION('Ю', 2), RELATION('Я', 2), \  RELATION('а', 2), RELATION('б', 2), RELATION('в', 2), RELATION('г', 2), RELATION('д', 2), RELATION('е', 2), \  RELATION('ё', 2), RELATION('ж', 2), RELATION('з', 2), RELATION('и', 2), RELATION('й', 2), RELATION('к', 2), \  RELATION('л', 2), RELATION('м', 2), RELATION('н', 2), RELATION('о', 2), RELATION('п', 2), RELATION('р', 2), \  RELATION('с', 2), RELATION('т', 2), RELATION('у', 2), RELATION('ф', 2), RELATION('х', 2), RELATION('ц', 2), \  RELATION('ч', 2), RELATION('ш', 2), RELATION('щ', 2), RELATION('ъ', 2), RELATION('ы', 2), RELATION('ь', 2), \  RELATION('э', 2), RELATION('ю', 2), RELATION('я', 2), RELATION(' ', 2), RELATION('.', 2), RELATION(',', 2), \  RELATION('?', 2), RELATION('!', 2), RELATION(';', 2), RELATION(':', 2), RELATION('-', 2), RELATION(')', 2), \  RELATION('(', 2)), \  \  NODE(1, RELATION('"', 3)), \  NODE()  );  FST fstIf(LEX\_IF, SYMB\_LEXEMS, "", 3, NODE(1, RELATION('i', 1)),  NODE(1, RELATION('f', 2)),  NODE()  );  FST fstNext(LEX\_NEXT, SYMB\_LEXEMS, "", 5, NODE(1, RELATION('n', 1)), \  NODE(1, RELATION('e', 2)), \  NODE(1, RELATION('x', 3)), \  NODE(1, RELATION('t', 4)), \  NODE()  );  FST fstElse(LEX\_ELSE, SYMB\_LEXEMS, "", 5, NODE(1, RELATION('e', 1)), \  NODE(1, RELATION('l', 2)), \  NODE(1, RELATION('s', 3)), \  NODE(1, RELATION('e', 4)), \  NODE()  );  FST fstEscape(LEX\_ESCAPE, SYMB\_STATIC\_LIB, "", 7, NODE(1, RELATION('e', 1)), \  NODE(1, RELATION('s', 2)), \  NODE(1, RELATION('c', 3)), \  NODE(1, RELATION('a', 4)), \  NODE(1, RELATION('p', 5)), \  NODE(1, RELATION('e', 6)), \  NODE()  );  FST fstSubline(LEX\_SUBLINE, SYMB\_STATIC\_LIB, "", 8, NODE(1, RELATION('s', 1)), \  NODE(1, RELATION('u', 2)), \  NODE(1, RELATION('b', 3)), \  NODE(1, RELATION('l', 4)), \  NODE(1, RELATION('i', 5)), \  NODE(1, RELATION('n', 6)), \  NODE(1, RELATION('e', 7)), \  NODE()  );  FST fstLength(LEX\_LENGTH, SYMB\_STATIC\_LIB, "", 7, NODE(1, RELATION('l', 1)), \  NODE(1, RELATION('e', 2)), \  NODE(1, RELATION('n', 3)), \  NODE(1, RELATION('g', 4)), \  NODE(1, RELATION('t', 5)), \  NODE(1, RELATION('h', 6)), \  NODE()  );  } |

|  |
| --- |
| struct Entry  {  char lexeme; // лексема  int stringNumber; // номер строки в исходном коде  int idxTI; // индекс в таблице идентификаторов  short priority; // приоритет для операций  };  struct LexTable  {  int maxsize; // емкость таблицы лексем  int size; // текущий размер таблицы лексем  Entry\* table; // массив строк таблицы лексем  }; |