МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра программной инженерии

Специальность 6-05-0612-01 Программная инженерия

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора ZEO – 2024»

Выполнил студент Закревский Егор Олегович

(Ф.И.О.)

Руководитель проекта преп.-стажер Некрасова А.П.

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

Заведующий кафедрой к.т.н., доц. Смелов В.В

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

Консультанты преп.-стажер Некрасова А.П.

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

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

Нормоконтролер преп.-стажер Некрасова А.П

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

Курсовой проект защищен с оценкой

Минск 2024

Содержание

[Введение 7](#_Toc185535597)

[1 Спецификация языка программирования 8](#_Toc185535598)

[1.1 Характеристика языка программирования 8](#_Toc185535599)

[1.2 Определение алфавита языка программирования 8](#_Toc185535600)

[1.3 Применяемые сепараторы 8](#_Toc185535601)

[1.4 Применяемые кодировки 9](#_Toc185535602)

[1.5 Типы данных 9](#_Toc185535603)

[1.6 Преобразование типов данных 9](#_Toc185535604)

[1.7 Идентификаторы 10](#_Toc185535605)

[1.8 Литералы 10](#_Toc185535606)

[1.9 Объявление данных 11](#_Toc185535607)

[1.10 Инициализация данных 11](#_Toc185535608)

[1.11 Инструкции языка 11](#_Toc185535609)

[1.12 Операции языка 12](#_Toc185535610)

[1.13 Выражения и их вычисление 12](#_Toc185535611)

[1.14 Конструкции языка 12](#_Toc185535612)

[1.15 Области видимости идентификаторов 13](#_Toc185535613)

[1.16 Семантические проверки 13](#_Toc185535614)

[1.17 Распределение оперативной памяти на этапе выполнения 14](#_Toc185535615)

[1.18 Стандартная библиотека и её состав 14](#_Toc185535616)

[1.19 Ввод и вывод данных 15](#_Toc185535617)

[1.20 Точка входа 15](#_Toc185535618)

[1.21 Препроцессор 15](#_Toc185535619)

[1.22 Соглашения о вызовах 15](#_Toc185535620)

[1.23 Объектный код 15](#_Toc185535621)

[1.24 Классификация сообщений транслятора 15](#_Toc185535622)

[1.25 Контрольный пример 16](#_Toc185535623)

[2 Структура транслятора 17](#_Toc185535624)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 17](#_Toc185535625)

[2.3 Протоколы, формируемые транслятором 18](#_Toc185535626)

[3 Разработка лексического анализатора 19](#_Toc185535627)

[3.1 Структура лексического анализатора 19](#_Toc185535628)

[3.2 Контроль входных символов 19](#_Toc185535629)

[3.3 Удаление избыточных символов 20](#_Toc185535630)

[3.4 Перечень ключевых слов 20](#_Toc185535631)

[3.5 Основные структуры данных 22](#_Toc185535632)

[3.6 Структура и перечень сообщений лексического анализатора 23](#_Toc185535633)

[3.7 Принцип обработки ошибок 23](#_Toc185535634)

[3.8 Параметры лексического анализатора и режимы его работы 23](#_Toc185535635)

[3.9 Алгоритм лексического анализа 23](#_Toc185535636)

[3.10 Контрольный пример 25](#_Toc185535637)

[4. Разработка синтаксического анализатора 26](#_Toc185535638)

[4.1 Структура синтаксического анализатора 26](#_Toc185535639)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 26](#_Toc185535640)

[4.3 Построение конечного магазинного автомата 27](#_Toc185535641)

[4.4 Основные структуры данных 28](#_Toc185535642)

[4.5 Описание алгоритма синтаксического разбора 28](#_Toc185535643)

[4.6 Структура и перечень сообщений синтаксического анализатора 28](#_Toc185535644)

[4.7 Параметры синтаксического анализатора 28](#_Toc185535645)

[4.8. Принцип обработки ошибок 29](#_Toc185535646)

[4.9 Контрольный пример 29](#_Toc185535647)

[5 Разработка семантического анализатора 30](#_Toc185535648)

[5.1 Структура семантического анализатора 30](#_Toc185535649)

[5.2 Функции семантического анализатора 30](#_Toc185535650)

[5.3 Структура и перечень сообщений семантического анализатора 30](#_Toc185535651)

[5.4 Принцип обработки ошибок 31](#_Toc185535652)

[5.5 Контрольный пример 31](#_Toc185535653)

[6. Преобразование выражений 32](#_Toc185535654)

[6.1 Выражения, допускаемые языком 32](#_Toc185535655)

[6.2 Польская запись и принцип её построения 32](#_Toc185535656)

[6.3 Программная реализация обработки выражений 33](#_Toc185535657)

[6.4 Контрольный пример 33](#_Toc185535658)

[7. Генерация кода 34](#_Toc185535659)

[7.1 Структура генератора кода 34](#_Toc185535660)

[7.2 Представление типов данных в оперативной памяти 34](#_Toc185535661)

[7.3 Статическая библиотека 35](#_Toc185535662)

[7.4 Особенности алгоритма генерации кода 35](#_Toc185535663)

[7.5 Параметры, управляющие генерацией кода 36](#_Toc185535664)

[7.6 Контрольный пример 36](#_Toc185535665)

[8 Тестирование транслятора 37](#_Toc185535666)

[8.1 Тестирование проверки на допустимость символов 37](#_Toc185535667)

[8.2 Тестирование лексического анализатора 37](#_Toc185535668)

[8.3 Тестирование синтаксического анализатора 38](#_Toc185535669)

[8.4 Тестирование семантического анализатора 38](#_Toc185535670)

[Заключение 40](#_Toc185535671)

[Список использованных источников 41](#_Toc185535672)

[ПРИЛОЖЕНИЕ А 42](#_Toc185535673)

[ПРИЛОЖЕНИЕ Б 44](#_Toc185535674)

[ПРИЛОЖЕНИЕ В 45](#_Toc185535675)

[ПРИЛОЖЕНИЕ Г 63](#_Toc185535676)

[ПРИЛОЖЕНИЕ Д 66](#_Toc185535677)

[ПРИЛОЖЕНИЕ Е 69](#_Toc185535678)

[ПРИЛОЖЕНИЕ Ж 74](#_Toc185535679)

# **Введение**

Задачей данного курсового проекта была поставлена разработка транслятора своего языка программирования ZEO-2024. Он предназначен для выполнения арифметических и логических действий над числами.

Главная идея транслятора заключается в том, чтобы сделать исходный код на данном языке программирования понятным компьютеру. Для решения этой задачи был выбран способ трансляции исходного в исходный код на языке JavaScript.

Исходя из цели курсового проекта, были определены следующие задачи:

* разработка спецификации языка программирования;
* разработка структуры транслятора;
* разработка лексического анализатора;
* разработка синтаксического анализатора;
* разработка семантического анализатора;
* обработка выражений с помощью польской инверсии;
* генерация кода на язык JavaScript;
* тестирование транслятора.

Способы решения каждой задачи будут описаны в соответствующих главах курсового проекта.

В первой главе работы определена спецификация языка программирования.

Во второй главе представлена структура транслятора. В ней перечислены компоненты транслятора, их назначения и принципы взаимодействия.

В третьей главе описана разработка лексического анализатора, который создаёт таблицы лексем и идентификаторов.

В четвертой главе описана разработка синтаксического анализатора, который выполняет разбор исходного кода в соответствии с правилами языка программирования.

В пятой главе описан семантический анализатор, которые проверяет исходный код программы на наличие семантических ошибок.

В шестой главе описан способ преобразования выражений в польский формат.

В седьмой главе представлена генерация кода в язык JavaScript с помощью таблиц лексем и идентификаторов.

В восьмой главе описывается тестирование транслятора.

**1 Спецификация языка программирования**

**1.1 Характеристика языка программирования**

Язык программирования ZEO-2024 – это универсальный язык высокого уровня. Он является процедурным, компилируемым, не объектно-ориентированным. Язык строго типизируемый, что говорит о невозможности преобразования типов, транслируемым языком программирования.

**1.2 Определение алфавита языка программирования**

Совокупность символов, используемых в языке, называется алфавитом языка.

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

**1.3 Применяемые сепараторы**

Символы-сепараторы служат в качестве разделителей операций языка. Сепараторы, используемые в языке программирования ZEO-2024, приведены в таблице 1.1.

Таблица 1.1 – Сепараторы

|  |  |  |
| --- | --- | --- |
| Сепаратор | Название | Область применения |
| ‘ ‘ | Пробел | Допускается везде, кроме идентификаторов и ключевых слов |
| ; | Точка с запятой | Разделение конструкций |
| {…} | Фигурные скобки | Заключение программного блока |
| […] | Квадратные кавычки | Блок кода |
| (…) | Круглые скобки | Приоритет операций, параметры функции |
| "…" | Двойные кавычки | Строковый литерал |
| '…' | Одинарные кавычки | Допускается везде, кроме идентификаторов и ключевых слов |
| = | Знак «равно» | Присваивание значения |
| , | Запятая | Разделение параметров |
| /  \  +  -  \*  :  %  &  | ~ | Знаки «косая черта», «обратная косая черта», «плюс», «минус», «звездочка», «двоеточие», «процент», «амперсанд», «вертикальная линия»,  «тильда» | Выражения |

**1.4 Применяемые кодировки**

Для написания исходного кода на языке программирования ZEO-2024 используется кодировка Windows-1251.Таблица символов Windows – 1251 представлена на рисунке 1.1



Рисунок 1.1 – Таблица символов кодировки Windows - 1251

**1.5 Типы данных**

В языке ZEO-2024 реализованы несколько типов данных. Тип данных представляет собой набор или группировку значений данных, обычно задаваемых набором возможных значений, набором разрешенных операций над этими значениями и/или представлением этих значений в виде типов компьютеров.

Типы данных, которые реализованы в языке программирования ZEO-2024:

- целочисленный (1 байт);

- символьный;

- логический;

- строковый.

**1.6 Преобразование типов данных**

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

**1.7 Идентификаторы**

Идентификаторы – имена, задаваемые в программе для переменных, типов и функций. Идентификаторы не должны совпадать с ключевыми словами. Имя идентификатора создается по следующим правилам:

* состоит из символов [a-z], [A-Z], [0-9] и ‘\_’;
* длина идентификаторов не должна превышать 10 символов.

При привышении максимального размера идентификатора, они усекаются. Имя идентификатора не может совпадать с именем функции, уже содержащаяся в стандартной библиотеке, если только это функция подключена через оператор add.

**1.8 Литералы**

Литералы – неизменяемые значения, или константы. Являются значением переменных. В языке программирования ZEO-2024 реализованы строковые, целочисленные, логические, символьные литералы. Численные литералы не требуют особого обозначения, логические определяются ключевыми словами «true» и «false», а символьные и строковые выделяются кавычками.

Таблица 1.2 – Описание литералов

|  |  |  |  |
| --- | --- | --- | --- |
| Тип литерала | Регулярное выражение | Описание | Пример |
| Целочисленный литерал | [1-9]+[0-9] | Целочисленные отрицательные литералы. Диапазон значений от -128 до 127 | declare byte var1;  var1 = 6;  6 – целочисленный литерал 1 байт. |
| Логический  Литерал | [false|true] | Принимает значение 0 и 1. Используются для указания истинности | declare bool var2;  var2 = true;  true – логический литерал |
| Символьный литерал | ‘[a-z|A-Z|0-9]’ | Символ, заключенный в ‘’ (одинарные кавычки), | declare char symbol;  symbol = ‘e’;  e – символьный литерал. |
| Строковый литерал | “[a-z|A-Z|0-9]” | Строковые литералы, максимальная длина строки 255 символов | write “text”  text – строковый литерал |

**1.9 Объявление данных**

Объявление переменных в языке ZEO-2024 происходит помощью ключевого слова «announce». объявлении предусмотрена неявная инициализация. Явная инициализация при объявлении запрещена. Все переменные должны находиться внутри программного блока. Можно объявлять переменные с одинаковым идентификатором в различных функциях, потому что каждая переменная получает префикс – название функции, в которой она объявлена. Глобальные переменные не предусмотрены языком.

**1.10 Инициализация данных**

При объявлении переменной не допускается инициализация. Описание способов инициализации переменных языка ZEO-2024 представлено в таблице 1.3.

Таблица 1.3 – Способы инициализации переменных

|  |  |  |
| --- | --- | --- |
| Конструкция | Описание | Пример |
| announce <тип данных> <идентификатор>; | Неявная инициализация: переменные типа byte инициализируются нулем, переменные типа char и string – пустым символом, переменные типа bool – значением «0». | announce byte sum;  announce char chr;  announce strg str;  announce bool bl; |
| <идентификатор> = <значение>; | Присваивание переменной значения. | sum = 9;  chr = ‘D’; |

Соответствие типов проверяется на синтаксическом анализе.

**1.11 Инструкции языка**

Все возможные инструкции языка программирования ZEO-2024 представлены в общем виде в таблице 1.4.

Таблица 1.4 – Инструкции языка программирования ZEO-2024

|  |  |
| --- | --- |
| Инструкция | Запись на языке ZEO-2024 |
| Объявление переменной | announce <тип данных> <идентификатор>; |
| Объявление функции | announce <тип данных> fnct <идентификатор> (<тип данных> <идентификатор>, …) {<блок кода>}; |
| Возврат значения | return <литерал>|<идентификатор>; |
| Вывод | write <идентификатор>; |

Продолжение таблицы -1.4 – Инструкции языка программирования ZEO-2024

|  |  |
| --- | --- |
| Вывод с переносом на следующую строку | writeline <идентификатор>; |
| Присваивание | <идентификатор>=<значение>/<идентификатор>/<функция>; |

**1.12 Операции языка**

Язык программирования ZEO-2024 может выполнять операции, представленные в таблице 1.5.

Таблица 1.5 – Операции языка программирования ZEO-2024

|  |  |  |  |
| --- | --- | --- | --- |
| Операция | Примечание | Типы данных | Пример |
| (  ) | Приоритет операций |  | sum = (a + b) \* c; |
| = | Присваивание | (byte, byte)  (char, char)  (string, string)  (bool, bool) | a = 33;  ch = ‘H’;  str = “Hello”;  logic = false; |
| <,> | Знаки «больше», «меньше» для условной инструкции | (byte, byte)  (char, char) | whilst(sum < diff) […]; |
| ! | Оператор неравенства | (byte, byte)  (char, char) | whilst(sum ! diff) […]; |
| & | Логическое и | (byte, byte) | and = a & b; |
| | | Логическое или | (byte, byte) | or = a | b; |
| ~ | Инверсия | (byte) | inv = ~a; |
| + | Суммирование | (byte, byte) | sum = a + b; |
| - | Вычитание | (byte, byte) | diff = a – b; |
| \* | Умножение | (byte, byte) | mul = a\*b; |
| : | Деление | (byte, byte) | div = a:b; |
| % | Остаток от деления | (byte, byte) | mod = a % b; |
| / | Сдвиг вправо | (byte, byte) | sdv = a / b; |
| \ | Сдвиг влево | (byte, byte) | sdv = a \ b; |

**1.13 Выражения и их вычисление**

Круглые скобки в выражении используются для изменения приоритета операций. Также не допускается запись двух подряд идущих арифметических операций. Выражения вычисляются только после оператора присваивания.

## **1.14 Конструкции языка**

Ключевые программные конструкции языка программирования ZEO-2024 представлены в таблице 1.6.

Таблица 1.6 – Программные конструкции языка ZEO-2024

|  |  |
| --- | --- |
| Конструкция | Запись на языке ZEO-2024 |
| Главная функция (точка входа) | main  {  …  return <идентификатор> / <литерал>;  } |
| Функция | announce <тип данных> fnct <идентификатор> (<тип данных> <идентификатор>, …)  {… return <идентификатор> / <литерал>; }; |
| Цикл | whilst(a > 8)[ …]; |
| Условный оператор | if(a > 4)[…]; |

Программные конструкции языка ZEO-2024 представляют собой базовый функционал для выполнения различных операций, что делает возможным решать задачи различного уровня.

**1.15 Области видимости идентификаторов**

В языке ZEO-2024 все переменные являются локальными, т.е. имеют функциональную область видимости. Они обязаны находится внутри программного блока функций (по принципу С++). Объявление глобальных переменных не предусмотрено. Объявление пользовательских областей видимости не предусмотрено.

**1.16 Семантические проверки**

Назначение семантического анализа – проверка смысловой правильности конструкций языка программирования. Таблица с перечнем семантических проверок, предусмотренных языком, приведена в таблице 1.7.

Таблица 1.7 – Семантические проверки

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Идентификаторы не должны повторно объявляться в пределах одной функции. |
| 2 | Тип возвращаемого значения должен совпадать с типом функции при её объявлении или подключении |

Продолжение таблицы 1.7 – Семантические проверки

|  |  |
| --- | --- |
| 3 | Тип данных передаваемых значений в функцию должен совпадать с типом параметров при её объявлении или подключении |
| 4 | В функцию должно быть передано то число параметров, сколько ожидается |
| 5 | Тип данных результата выражения должен совпадать с типом данных идентификатора, которому оно присваивается |
| 6 | Типы данных операндов выражения должны быть одинаковыми |
| 7 | Тип данных string не может быть аргументом условной конструкции |
| 8 | Для типа char определены только операции + и - |
| 9 | Функции не должны подключаться дважды в пределах одной программы |

Если семантическая проверка не проходит, то в лог журнал записывается соответствующая ошибка.

**1.17 Распределение оперативной памяти на этапе выполнения**

Транслированный код использует две области памяти. В сегмент констант заносятся все литералы. В сегмент данных заносятся переменные и параметры функций. Локальная область видимости в исходном коде определяется за счет использования правил именования идентификаторов и регулируется их префиксами, что и обуславливает их локальность на уровне исходного кода, несмотря на то, что в оттранслированным в язык ассемблера коде переменные имеют глобальную область видимости.

**1.18 Стандартная библиотека и её состав**

Стандартная библиотека ZEO-2024 написана на языке программирования C++.

Для использования функций стандартной библиотеки, нужно явно подключить необходимую функцию с помощью ключевого слова add, далее работа с ними производится как с пользовательскими функциями. Функции стандартной библиотеки с описанием представлены в таблице 1.8.

Таблица 1.8 – Состав стандартной библиотеки

|  |  |  |
| --- | --- | --- |
| Функция(C++) | Возвращаемое значение | Описание |

Продолжение таблицы 1.8 – Состав стандартной библиотеки

|  |  |  |
| --- | --- | --- |
| compare(str1,str2) | byte | Возвращает результат сравнения двух строк |
| absolute(byte) | byte | Возвращает модуль числа |

**1.19 Ввод и вывод данных**

В языке ZEO-2024 не реализованы средства ввода данных.

Для вывода данных в стандартный поток вывода предусмотрен оператор write, который базируется на приватных функциях стандартной библиотеки, этот оператор выводит ваши данные. Оператор writeline начинает вывод с новой строки.

**1.20 Точка входа**

В языке ZEO-2024 каждая программа должна содержать главную функцию main, т. е. точку входа, с которой начнется последовательное выполнение программы.

**1.21 Препроцессор**

Препроцессор в языке программирования ZEO-2024 не предусмотрен.

**1.22 Соглашения о вызовах**

Так как компиляция идет в JavaScript, то соглашение о вызовах в языке ZEO-2024 отсутствует.

**1.23 Объектный код**

Язык программирования ZEO-2024 транслируется в язык JavaScript.

**1.24 Классификация сообщений транслятора**

В случае возникновения ошибки в коде программы на языке ZEO-2024 и выявления её транслятором в текущий файл протокола выводится сообщение. Классификация сообщений приведена в таблице 1.9.

Таблица 1.9. – Классификация сообщений транслятора

|  |  |
| --- | --- |
| Интервал | Описание ошибок |
| 0-99 | Системные ошибки |
| 100-109 | Ошибки параметров |
| 110-119 | Ошибки открытия и чтения файлов |
| 120-129 | Ошибки лексического анализа |
| 130-139 | Ошибки таблиц лексем и таблиц идентификаторов |
| 600-699 | Ошибки синтаксического анализа |
| 700-900 | Ошибки семантического анализа |

Компилятор может обрабатывать до 900 различных ошибок.

**1.25 Контрольный пример**

Контрольный пример демонстрирует главные особенности языка ZEO-2024: его фундаментальные типы, основные структуры, функции, процедуры, использование функций статической библиотеки. Исходный код контрольного примера представлен в приложении А.

# **2 Структура транслятора**

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

Транслятор преобразует программу, написанную на языке ZEO-2024 в программу на языке JavaScript [1][2]. Для указания выходных файлов используются входные параметры транслятора, Компонентами транслятора являются лексический, синтаксический и семантический анализаторы, а также генератор кода на язык JavaScript.

Лексический анализ – первая фаза трансляции. Назначением лексического анализатора является нахождение ошибок лексики языка ZEO-2024 и формирование таблицы лексем и таблицы идентификаторов.

Семантический анализ в свою очередь является проверкой исходной программы ZEO-2024 на семантическую согласованность с определением языка, т.е. проверяет правильность текста исходной программы с точки зрения семантики.

Синтаксический анализ – это основная часть транслятора, предназначенная для распознавания синтаксических конструкций и формирования промежуточного кода ZEO-2024. Для этого используются таблица лексем и идентификаторов. Синтаксический анализатор распознаёт синтаксические конструкции, выявляет синтаксические ошибки при их наличии и формирует дерево разбора

Генератор кода – этап транслятора, выполняющий генерацию JavaScript кода на основе полученных данных на предыдущих этапах трансляции. Генератор кода принимает на вход таблицы идентификаторов и лексем и транслирует код на языке ZEO-2024, прошедший успешно все предыдущие этапы, в код на языке JavaScript.

**2.2 Перечень параметров транслятора**

Входные параметры представлены в таблице 2.1.

Таблица 2.1 – Входные параметры транслятора языка ZEO-2024

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<имя\_файла> | Входной файл с любым расширением, в котором содержится исходный код на языке ZEO-2024. Данный параметр должен быть указан обязательно. В случае если он не будет задан, не будет | Не предусмотрено |
| -log:<имя\_файла> | Файл содержит в себе краткую информацию об исходном коде на языке ZEO-2024. В этот файл могут быть выведены таблицы идентификаторов, лексем, а также дерево разбора. | <имя\_файла>.log |
| -out:<имя\_файла> | В этот файл будет записан результат трансляции кода на язык JavaScript | <имя\_файла>.asm |
| tree | Вывод дерева разбора синтаксического анализатора. | false |
| lex | Вывод таблицы лексем | false |
| id | Вывод таблицы идентификаторов | false |

Как видно из таблицы, транслятор языка ZEO-2024 имеет 6 входных параметра. Если не указать параметры -log или -out, то по умолчанию они будут созданы с названием файла, указанного в параметре -in.

**2.3 Протоколы, формируемые транслятором**

Таблица с перечнем протоколов, формируемых транслятором языка ZEO-2024 и их назначением представлена в таблице 2.2.

Таблица 2.2 – Протоколы, формируемые транслятором языка ZEO-2024

|  |  |
| --- | --- |
| Формируемый протокол | Описание протокола |
| Файл журнала, “\*.log ” | Файл содержит в себе краткую информацию об исходном коде на языке ZEO-2024. В этот файл выводится протокол работы анализаторов, а так же различные ошибки |
| “\*.asm” | Содержит сгенерированный код на языке Ассемблера. |

В log файл выводятся все ошибки, за исключением тех, что связаны с открытием файла log или считывания параметров.

# **3 Разработка лексического анализатора**

**3.1 Структура лексического анализатора**

Лексический анализатор – часть транслятора, выполняющая лексический анализ. Лексический анализатор принимает обработанный и разбитый на отдельные компоненты исходный код на языке ZEO-2024. Структура лексического анализатора представлена на рисунке 3.1.



Рисунок 3.1 – Структура лексического анализатора

Результатом работы лексического анализатора являются заполненные таблица лексем и таблица идентификаторов.

**3.2 Контроль входных символов**

Для удобной работы с исходным кодом, при передаче его в лексический анализатор, все символы разделяются по категориям. Таблица входных символов представлена в приложении Е.

В языке ZEO – 2024 присутствуют определенные виды символов. Входным символам соответствует иные символы, предназначенные для синтаксического анализа. Соответствие символов и их значений представлено в таблице 3.1.

Таблица 3.1 - Соответствие символов и их значений в таблице

|  |  |
| --- | --- |
| Значение в таблице входных символов | Символы |
| Разрешенный | T |
| Запрещенный | F |

Продолжение таблицы 3.1 - Соответствие символов и их значений в таблице

|  |  |
| --- | --- |
| Символ двойной кавычки | D |
| Пробел, табуляция | S |
| Символ начала комментария | O |
| Символ новой строки | N |
| Символ-разделитель или оператор | L |
| Символ одинарные кавычки | C |

Если символ во входном файле отмечен в таблице контроля, тогда лексический анализатор выдаст соответствующую ошибку.

**3.3 Удаление избыточных символов**

Удаление избыточных символов не предусмотрено, так как после проверки на допустимость символов исходный код на языке программирования ZEO-2024 разбивается на токены, которые записываются в очередь.

## **3.4 Перечень ключевых слов**

Лексический анализатор преобразует исходный текст, заменяя лексические единицы лексемами для создания промежуточного представления исходной программы. Соответствие токенов и лексем приведено в таблице 3.2.

Таблица 3.2 – Соответствие токенов и сепараторов с лексемами

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Пояснение |
| byte, strg,char,bool | t | Названия типов данных языка. |
| Идентификатор | i | Длина идентификатора – 10 символов. |
| Литерал | l | Литерал любого доступного типа. |
| fnct | f | Объявление функции. |
| announce | d | Объявление переменной |
| return | r | Выход из функции/процедуры. |

Продолжение таблицы 3.2 – Соответствие токенов и сепараторов с лексемами

|  |  |  |
| --- | --- | --- |
| main | m | Главная функция. |
| write | u | Вывод данных |
| ; | ; | Сепаратор |
| , | , | Сепаратор |
| { | { | Начало блока/тела функции. |
| } | } | Закрытие блока/тела функции. |
| ( | ( | Передача параметров в функцию, приоритет операций. |
| ) | ) | Закрытие блока для передачи параметров, приоритет операций. |
| = | v | Оператор присваивания. |
| +  -  \*  /  % | v  v  v  v  v | Знаки операций |
| >  <  ! | b  b  b | Знаки логических операторов. |

Каждое выражение соответствует детерминированному конечному автомату, который представляет собой автомат с конечными состояниями, выполняющий разбор данного выражения. Для каждого автомата из массива на вход подается строка, и с помощью регулярного выражения, описывающего переходы между состояниями данного автомата, выполняется разбор. Если разбор прошел успешно, выражение записывается в таблицу лексем. В случае, если выражение является идентификатором или литералом, соответствующая информация дополнительно заносится в таблицу идентификаторов.

Пример реализации таблицы лексем и конечные автоматы, соответствующие лексемам языка ZEO-2024, приведены в приложении В.

Структура конечного автомата изображена на листинге 3.1

struct libfuncs {

std::string name;

IT::IDDATATYPE params[255];

int parcount = 0;

IT::IDDATATYPE retval;

};

FST::RELATION::RELATION(char c, short ns)

{

symbol = c;

nnode = ns;

}

FST::NODE::NODE()

{

n\_relation = 0;

RELATION\* relations = NULL;

};

FST::NODE::NODE(short n, RELATION rel, ...)

{

n\_relation = n;

RELATION\* p = &rel;

relations = new RELATION[n];

for (short i = 0; i < n; i++)

relations[i] = p[i];

};

FST::FST::FST(char\* s, short ns, NODE n, ...)

{

string = s;

nstates = ns;

nodes = new NODE[ns];

NODE\* p = &n;

for (int k = 0; k < ns; k++)

nodes[k] = p[k];

rstates = new short[nstates];

rstates[0] = 0;

position = -1;

};

листинг 3.1 – Структура конечного автомата

## **3.5 Основные структуры данных**

Основные структуры таблиц лексем и идентификаторов языка ZEO-2024, используемые для хранения данных, описаны в приложении В. Таблица лексем включает следующие поля: лексема, ее порядковый номер, присвоенный при разборе, номер строки и столбца в исходном коде, индекс в таблице идентификаторов (если идентификатор отсутствует, индекс принимает значение -1), а также поле, содержащее значение лексемы. Таблица идентификаторов содержит имя идентификатора, его номер в таблице лексем, тип данных, тип идентификатора, значение, а также бинарное поле, указывающее, является ли идентификатор внешним.

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

Индексы ошибок, обнаруживаемых лексическим анализатором, находятся в диапазоне 120-125. Перечень сообщений лексического анализатора представлен в таблице 3.3.

Таблица 3.3 – ошибки, определяемые синтаксическим анализатором

|  |  |
| --- | --- |
| Номер ошибки | Сообщение об ошибке |
| 120 | [LA]: Ошибка при разборе токена |
| 121 | [LA]: Используется необъявленный идентификатор |
| 122 | [LA]: Идентификатор не имеет типа |
| 124 | [LA]: Отсутствует точка входа |
| 125 | [LA]: Обнаружена вторая точка входа |

Также сам текст ошибки содержит в себе префикс [LA].

**3.7 Принцип обработки ошибок**

Все ошибки являются критическими и приводят к прекращению работы транслятора и выводу диагностического сообщения в log-файл.

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

Входные параметры используются для вывода результата работы лексического анализатора. Они передаются аргументами через командную строку. Данные параметры были рассмотрены в таблице 2.1.

## **3.9 Алгоритм лексического анализа**

Алгоритм лексического анализа заключается в последовательном распознавании и разборе последовательностей исходного кода с заполнением таблиц лексем и идентификаторов. Лексический анализатор выполняет обработку цепочек текста программы, используя конечные автоматы, представленные в виде графов. Если подходящий автомат не найден, фиксируется номер строки, где расположен токен, и выводится сообщение об ошибке. В случае успешной обработки токена дальнейшие действия зависят от его типа. Регулярные выражения представляют собой формальный способ задания регулярных языков, основанный на использовании констант и операторов для определения множества строк и операций над ними. Любое регулярное выражение можно преобразовать в граф.

Если токен является идентификатором, перед его именем указывается название функции, в которой он объявлен, и он добавляется в таблицу идентификаторов. Для идентификатора функции название функции, где он объявлен, не записывается. Если токен представляет собой литерал, он заносится в таблицу идентификаторов под именем вида «abi», где «a» — имя функции, в которой объявлен литерал, «b» — «$LEXEM», а «c» — порядковый номер литерала, увеличенный на единицу.

При встрече ключевого слова, обозначающего тип данных или вид идентификатора, соответствующая лексема записывается в таблицу лексем, а также сохраняется информация о типе данных или виде идентификатора. Когда в дальнейшем встречается идентификатор, он добавляется в таблицу идентификаторов с указанным типом данных, видом идентификатора и именем вида «ab», где «a» — имя функции, в которой объявлен идентификатор, а «b» — имя самого идентификатора. Конечный автомат, соответствующий ключевому слову «main», приведен на листинге 3.2.

FST l\_main(

str,

5,

NODE(1, RELATION('m', 1)),

NODE(1, RELATION('a', 2)),

NODE(1, RELATION('i', 3)),

NODE(1, RELATION('n', 4)),

NODE()

);

Листинг 3.2 – конечный автомат для ключевого слова «main»

В структуре «FST» узел графа обозначается как «NODE». Этот узел содержит информацию о количестве рёбер и сами рёбра, которые описаны в структуре «RELATION». В этой структуре указывается символ, определяющий переход, и номер узла, куда осуществляется этот переход.

После этого вызывается функция «execute», которая реализует разбор цепочки с использованием метода двух массивов в соответствии с заданным конечным автоматом. Реализация метода включает создание массива с предварительным выделением памяти под него, после чего в цикле выполняются шаги алгоритма, соответствующие разбору графа конечного автомата. Результат выполнения разбора определяет возвращаемое значение: «true» — если разбор завершился успешно, или «false» — если произошла ошибка. Подробная реализация функции представлена на листинге 3.3.

bool FST::execute(FST& fst)

{

fst.rstates[0] = 0;

fst.position = -1;

colum.reset\_col();

short\* rstates = new short[fst.nstates];

memset(rstates, 0xff, sizeof(short) \* fst.nstates);

short lstring = strlen(fst.string);

bool rc = true;

for (short i = 0; i < lstring && rc; i++)

{

fst.position++;

rc = step(fst, rstates, lstring);

colum.set\_col(i);

}

delete[] rstates;

return (rc ? (fst.rstates[fst.nstates - 1] == lstring) : rc);

}

Листинге 3.3 – реализация функции «execute»

В случае успешного разбора цепочки, в зависимости от лексемы, соответствующей конечному автомату, выполняются различные действия, в случае ключевого слова «main» будет вызван код, указанный на листинге 3.4.

case LEX\_MAIN: {

if (!wasMain) {

isRequireBodyFunc = true;

wasMain = true;

break;

}

else

throw ERROR\_THROW\_IN(125, lex.line, lex.col);

}

Листинг 3.4 – код, вызванный для ключевого слова «main»

В результате программа получит информацию о наличии точки входа и при появлении повторной вызовет соответствующую ошибку.

**3.10 Контрольный пример**

Результат работы лексического анализатора в виде таблиц лексем и идентификаторов, соответствующих контрольному примеру, представлен в приложении В.

**4. Разработка синтаксического анализатора**

**4.1 Структура синтаксического анализатора**

Синтаксический анализатор – это часть программы, которая выполняет процесс синтаксического анализа.

Синтаксический анализ – это фаза трансляции, выполняемая после лексического анализа и предназначенная для распознавания синтаксических конструкций ZEO-2024. Структура синтаксического анализатора представлена на рисунке 4.1.



Рисунок 4.1 – Структура синтаксического анализатора ZEO-2024

Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов. Выходной информацией – дерево разбора и протокол работы анализатора, который описывает процесс анализа.

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

В синтаксическом анализаторе транслятора языка ZEO-2024 используется контекстно-свободная грамматика, где

T – множество терминальных символов

N – множество нетерминальных символов

P – множество правил языка

S – начальный символ грамматики, являющийся нетерминалом.

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

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

Грамматика языка ZEO-2024 представлена в приложении Г.

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

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

Протокол и ошибки работы синтаксического анализатора выводятся в «log» журнал.

## **4.3 Построение конечного магазинного автомата**

Конечный автомат с магазинной памятью представляет собой семерку , описание которой представлено в таблице 4.1. Структура данного автомата показана в приложении Г.

Таблица 4.1 – Описание компонентов магазинного автомата

|  |  |  |
| --- | --- | --- |
| Компонент | Определение | Описание |
|  | Множество состояний автомата | Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата |
|  | Алфавит входных символов | Алфавит является множеством терминальных и нетерминальных символов, описание которых содержится ранее. |
|  | Алфавит специальных магазинных символов | Алфавит магазинных символов содержит стартовый символ и маркер дна стека |
|  | Функция переходов автомата | Функция представляет из себя множество правил грамматики. |
|  | Начальное состояние автомата | Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики (нетерминальный символ А) |
|  | Начальное состояние магазина автомата | Символ маркера дна стека ($) |
|  | Множество конечных состояний | Конечные состояние заставляют автомат прекратить свою работу. Конечным состоянием является пустой магазин автомата и совпадение позиции на входной ленте автомата с размером ленты |

Для вывода результата работы синтаксического анализатора нужно использовать флаг tree.

**4.4 Основные структуры данных**

Основные структуры данных синтаксического анализатора включают магазинный конечный автомат и грамматику Грейбах, которая задаёт правила языка ZEO-2024. Эти структуры приведены в приложении Г.

## **4.5 Описание алгоритма синтаксического разбора**

Принцип работы автомата следующий.

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

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

Перечень сообщений синтаксического анализатора представлен на листинге 4.1.

ERROR\_ENTRY(600, "[Syntaxis]: Неверная структура программы"),

ERROR\_ENTRY(601, "[Syntaxis]: Ошибочный оператор"),

ERROR\_ENTRY(602, "[Syntaxis]: Ошибка в выражении"),

ERROR\_ENTRY(603, "[Syntaxis]: Ошибка в параметрах функции"),

ERROR\_ENTRY(604, "[Syntaxis]: Ошибка в параметрах вызываемой функции"),

ERROR\_ENTRY(605, "[Syntaxis]: Ошибка знака в выражении"),

ERROR\_ENTRY(606, "[Syntaxis]: Ошибка синтаксического анализа"),

ERROR\_ENTRY(607, "[Syntaxis]: Ошибка условной конструкции"),

ERROR\_ENTRY\_NODEF(608),

ERROR\_ENTRY(609, "[Syntaxis]: Обнаружена синтаксическая ошибка (смотри журнал Log)"),

Листинг 4.1 - Перечень сообщений синтаксического анализатора

## **4.7 Параметры синтаксического анализатора**

Для вывода результата работы синтаксического анализатора используются входные параметры, описанные в пункте 2.2.

**4.8. Принцип обработки ошибок**

Обработка ошибок происходит следующим образом:

1. Синтаксический анализатор перебирает все правила и цепочки правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем.
2. Если невозможно подобрать подходящую цепочку, то генерируется соответствующая ошибка.
3. В случае ошибки выводится соответствующее сообщение в журнал «log» и компилятор прекращает работу.

## **4.9 Контрольный пример**

Результаты работы лексического разбора, а именно дерево разбора и протокол работы автомата с магазинной памятью приведены в приложении В.

**5 Разработка семантического анализатора**

**5.1 Структура семантического анализатора**

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

## **5.2 Функции семантического анализатора**

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

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

Сообщения, формируемые семантическим анализатором, представлены на листинге 5.1.

ERROR\_ENTRY(700, "[Semantic]: Повторное объявление идентификатора"),

ERROR\_ENTRY(701, "[Semantic]: Ошибка в возвращаемом значении"),

ERROR\_ENTRY(702, "[Semantic]: Ошибка в передаваемых значениях в функции: количество параметров не совпадает"),

ERROR\_ENTRY(703, "[Semantic]: Ошибка в передаваемых значениях в функции: типы параметров не совпадают"),

ERROR\_ENTRY(704, "[Semantic]: Нарушены типы данных в выражении или условной конструкции"),

ERROR\_ENTRY(705, "[Semantic]: Ошибка экспорта: в библиотеке нет такой функции"),

ERROR\_ENTRY(706, "[Semantic]: Ошибка экспорта: неверные параметры"),

ERROR\_ENTRY(707, "[Semantic]: Ошибка экспорта: ошибочный тип возвращаемого значения"),

ERROR\_ENTRY(708, "[Semantic]: Для строк операторы запрещены"),

ERROR\_ENTRY(709, "[Semantic]: Ошибочные параметры условной конструкции: строки не могут быть параметрами условной конструкции"),

ERROR\_ENTRY(710, "[Semantic]: Ошибочный оператор: для типа char разрешены только операции + и -"),

ERROR\_ENTRY(711, "[Semantic]: Для логических переменных использование арифметических и побитовых операторов запрещено"),

Листинг 5.1 - Сообщения, формируемые семантическим анализатором

На этапе семантического анализа предусмотрено возникновение 13 возможных ошибок.

**5.4 Принцип обработки ошибок**

Для того чтобы транслятор завершил свою работу будет достаточно обнаружить хотя бы одну ошибку, после чего произойдет запись информации об ошибке в «log» файл.

## **5.5 Контрольный пример**

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

# **6. Преобразование выражений**

## **6.1 Выражения, допускаемые языком**

В языке ZEO-2024 допускаются выражения, применимые к целочисленным типам данных. В выражениях поддерживаются арифметические операции, такие как +, -, \*, :, /, \, %, ~, &, | и (), и вызовы функций как операнды арифметических выражений. Приоритет операций представлен в таблице 6.1.

Таблица 6.1 – Приоритет операций в языке ZEO-2024

|  |  |
| --- | --- |
| Операция | Значение приоритета |
| ( | 1 |
| ) | 1 |
| + | 2 |
| - | 2 |
| \* | 3 |
| : | 3 |
| % | 3 |
| / | 3 |
| \ | 3 |
| & | 3 |
| | | 3 |
| ~ | 4 |

Все операции имеют определенный приоритет, который будет указывать порядок выполнения операций. Это можно увидеть в таблице 6.1.

**6.2 Польская запись и принцип её построения**

В языке ZEO-2024 польская запись отсутствует, т.к. язык компилируется в JavaScript. Польская запись — это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок. Существует два типа польской записи: прямая и обратная, также известные как префиксная и постфиксная. Отличие их от классического, инфиксного способа заключается в том, что знаки операций пишутся не между, а, соответственно, до или после аргументов.

Алгоритм построения:

– исходная строка: выражение;

– результирующая строка: польская запись;

– стек: пустой;

– результирующая строка: польская запись;

– 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 которого 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.

## **6.3 Программная реализация обработки выражений**

В языке ZEO-2024 польская запись отсутствует, т.к. язык компилируется в JavaScript.

## **6.4 Контрольный пример**

Пример преобразования выражений к обратной польской записи представлен в таблице 6.2.

Таблица 6.2 – Преобразование выражений к ОПЗ

|  |  |  |
| --- | --- | --- |
| Исходная строка | Результирующая строка | Стек |
| c+~155\*(d|88) |  |  |
| +~155\*(d|88) | c |  |
| ~155\*(d|88) | c | + |
| 155\*(d|88) | c | +~ |
| \*(d|88) | c155 | +~ |
| (d|88) | c155~ | +\* |
| d|88) | c155~ | +\*( |
| |88) | c155~d | +\*( |
| 88) | c155~d | +\*(| |
| ) | c155~d88 | +\*(| |
|  | c155~d88| | +\* |
|  | c155~d88|\* | + |
|  | c155~d88|\*+ |  |

**7. Генерация кода**

**7.1 Структура генератора кода**

Код на языке ZEO-2024 генерируется в формате языка JavaScript, предназначенного для процессоров x86. За процесс генерации отвечает модуль «CodeGeneration», который инициирует выполнение функции «Generate». Эта функция последовательно выполняет разные операции для генерирования исходного кода в код JavaScript.

Структура генератора кода представлена на листинге 7.1

void Generate(LEX::LEX t, Out::OUT o) {

Translate(t.lextable, t.idtable, \*0.stream);

}

Листинг 7.1 – структура генератора кода

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

Соответствия между типами данных идентификаторов на языке ZEO-2024 и на языке JavaScript приведены в таблице 7.1

Таблица 7.1 – Соответствие типов данных языка ZEO-2024 и JavaScript.

|  |  |
| --- | --- |
| Тип идентификатора на языке ZEO-2024 | Тип идентификатора на языке Ассемблера |
| char | symbol |
| string | string |
| byte | number |
| bool | boolean |

## **7.3 Статическая библиотека**

В языке ZEO-2024 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++. Объявление функций статической библиотеки генерируется автоматически в коде JavaScript. Все функции представлены в таблице 7.2.

Таблица 7.2 – функции статической библиотеки языка ZEO-2024

|  |  |
| --- | --- |
| Функция | Описание |
| compare | Сравнивает две строки и возвращает значение взависимости от результата |
| absolute | Возвращает модуль числа |

Подключение библиотеки в языке JavaScript происходит с помощью директивы import на этапе генерации кода. Далее указываются названия функций из библиотеки.

**7.4 Особенности алгоритма генерации кода**

Особенностями алгоритма генерации кода являются:

1. Объявление всех констант до начала кода.
2. Проверка деления на 0.
3. Взятие всего кода в функции try и catch для отлавливания ошибок.

**7.5 Параметры, управляющие генерацией кода**

На вход генератору кода поступают таблицы лексем и идентификаторов исходного код программы на языке ZEO-2024. Результаты работы генератора кода выводятся в файл с расширением .mjs.

**7.6 Контрольный пример**

Сгенерированный код можно посмотреть в приложении Ж.

# **8 Тестирование транслятора**

# **8.1 Тестирование проверки на допустимость символов**

В основе тестов, которым подвергался транслятор, лежали использование недопустимых структур и выражений, попытка ввода недопустимых символов, использование заведомо приводящих к ошибке ситуаций.

При обнаружении компилятор прекращает свою работу и выводит в файл протокола соответствующее сообщение. Также вся информация, выведенная до встречи ошибки остаётся в файле протокола, чтобы можно было проследить за тем, что привело к ошибке

# **8.2 Тестирование лексического анализатора**

На этапе лексического анализа в языке ZEO-2024 могут возникнуть ошибки. Результаты тестирования лексического анализатора показаны в таблицы 8.1.

Таблица 8.1 – Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main {  test1  } | Ошибка на этапе лексического анализатора  Ошибка 121: [LA]: Используется необъявленный идентификатор, строка 14, столбец 2 |
| main {  announce test;  } | Ошибка на этапе лексического анализатора  Ошибка 122: [LA]: Идентификатор не имеет типа, строка 14, столбец 11 |
| announce byte fnct test(byte tet) {} | Ошибка на этапе лексического анализатора  Ошибка 124: [LA]: Отсутствует точка входа |
| main {}  main{  announce byte test  } | Ошибка на этапе лексического анализатора  Ошибка 125: [LA]: Обнаружена вторая точка входа, строка 11, столбец 1 |

# **8.3 Тестирование синтаксического анализатора**

На этапе синтаксического анализа в языке ZEO-2024 могут возникнуть ошибки. Результаты тестирования синтаксического анализатора показаны в таблице 8.2

Таблица 8.2 – Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| function main  { } | 600: строка 11,[Syntaxis]: Неверная структура программы |
| main  {  announce byte byte;  } | 601: строка 14,[Syntaxis]: Ошибочный оператор |
| main  {  announce byte test;  test = 1 +- 4;  } | 602: строка 15,[Syntaxis]: Ошибка в выражении |
| announce byte fnct (strg a) {} | 603: строка 8,[Syntaxis]: Ошибка в параметрах функции |
| announce byte test;  test = 1\*4; | Ошибка 609: [Syntaxis]: Обнаружена синтаксическая ошибка (смотри журнал Log) |

# **8.4 Тестирование семантического анализатора**

Итоги тестирования семантического анализатора на корректное обнаружение семантических ошибок приведены в таблице 8.3

Таблица 8.3 – Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  {  announce byte test;  announce strg test; } | Ошибка 700: [Semantic]: Повторное объявление идентификатора, строка 15, столбец 15 |
| announce strg test;  test = “vdjs” + “vfsd”; | Ошибка 708: [Semantic]: Для строк операторы запрещены, строка 14, столбец 16 |
| announce strg fnct name()  {  return 0;  } | Ошибка 701: [Semantic]: Ошибка в возвращаемом значении, строка 8, столбец 8 |
| announce strg fnct name(byte a) {}  announce strg test;  test = name(“fdfs”); | Ошибка 703: [Semantic]: Ошибка в передаваемых значениях в функции: типы параметров не совпадают, строка 15, столбец 19 |

# **Заключение**

В ходе выполнения курсовой работы был разработан транслятор и генератор кода для языка программирования ZEO-2024 со всеми необходимыми компонентами. Таким образом, были выполнены основные задачи данной курсовой работы:

Сформулирована спецификация языка ZEO-2024;

1. Разработаны конечные автоматы и важные алгоритмы на их основе для эффективной работы лексического анализатора;

1. Осуществлена программная реализация лексического анализатора, распознающего допустимые цепочки спроектированного языка;
2. Разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
3. Осуществлена программная реализация синтаксического анализатора;
4. Разработан семантический анализатор, осуществляющий проверку используемых инструкций на соответствие логическим правилам;
5. Разработан транслятор кода на язык JavaScript;
6. Проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка ZEO-2024 включает:

1. 4 типа данных;

1. Поддержка операторов вывода;
2. Возможность вызова 2 функций стандартной библиотеки;
3. Наличие 3 операторов сравнения и скобок, 6 арифметических операторов, а также 3 операторов логики и 2 оператора сдвигов.
4. Язык поддерживает функции, операторы цикла и условия;
5. Структурированная и классифицированная система для обработки ошибок пользователя.

Проделанная работа предоставила необходимое представление о структурах и процессах, использующихся при построении транслятора, а также основные различия и преимущества тех или иных средств трансляции.

# **Список использованных источников**

1. Вирт Н. Построение компиляторов/ Пер. с англ. Борисов Е. В., Чернышов Л. Н. – М.: ДМК Пресс, 2010. – 192 с.: ил.

2. Грис Д. Конструирование компиляторов для цифровых вычислительных машин.: Пер. с англ. — М.: Мир, 1975.

3. Костельцев А. В. Построение интерпретаторов и компиляторов. СПб: Наука и Техника, 2001. — 224 стр. с ил.

4. Пратт Т. Языки программирования: разработка и реализация. Пер. с англ. — М.: Мир, 1979.

5. Хантер Р. Проектирование и конструирование компиляторов/ Пер. с англ.: Предисл. В. М. Савинкова. — М.: Финансы и статистика, 1984. — 232 с., ил.

6. Хендрикс Д. Компилятор языка Си для микроЭВМ: Пер. с англ. — М.: радио и связь, 1989. — 240 с.: ил.

# **ПРИЛОЖЕНИЕ А**

announce strg fnct cons(byte b)

{

whilst(b > 1)

[

writeline "Hello, world";

b = b - 1;

];

return "Everything successfuly!";

};

main

{

announce strg res;

res = cons(5);

writeline "The result of function cons";

writeline res;

add byte fnct compare(strg a, strg b);

add byte fnct absolute(byte a);

announce byte resultCompare1;

resultCompare1 = compare("str1","str");

writeline "The result of comparing two string";

writeline resultCompare1;

announce byte comparement2;

comparement2 = compare("str", "str1");

writeline "The result of comparing two string";

writeline comparement2;

announce byte abs;

abs = 0 - 2;

announce byte do;

do = absolute(abs);

writeline "The result modul number";

writeline do;

announce byte absolutly;

absolutly = 1 - 21 \* 3 : 3 + absolute(abs);

writeline "Output negative number";

writeline absolutly;

announce byte mod;

mod = 5 % 2;

writeline "The result mod two number";

writeline mod;

announce byte absol;

absol = absolute(absolutly);

writeline "The result of the modul return function";

writeline absol;

announce byte var;

var = 10;

writeline "Output integer number";

writeline var;

announce strg var1;

var1 = "Hello, world";

writeline "Output string";

writeline var1;

announce bool var4;

if(var4)

[

writeline "This variable is false";

];

var4 = true;

writeline "Output result condition";

if(var4)

[

writeline "This variable is true";

];

announce byte var5;

var5 = 1|2&3;

writeline var5;

announce byte var6;

var6 = 1&~1;

writeline var6;

announce byte var7;

var7 = 8/2;

writeline "The result of shift left by 2";

writeline var7;

announce byte var8;

var8 = 8\1;

writeline "The result of shift right by 1";

writeline var8;

announce byte zero;

announce byte name;

name = 2;

zero = 1 : (name - name);

writeline zero;

return 0;

}

Листинг 1 – Контрольный пример

# **ПРИЛОЖЕНИЕ Б**

#define IN\_CODE\_TABLE {\

IN::T,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::S,IN::N,IN::F,IN::F,IN::T,IN::F,IN::F,\

IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,\

IN::S,IN::L,IN::D,IN::F,IN::O,IN::L,IN::L,IN::C,IN::L,IN::L,IN::L,IN::L,IN::L,IN::L,IN::F,IN::L,\

IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::L,IN::L,IN::L,IN::L,IN::L,IN::F,\

IN::F,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,\

IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::L,IN::L,IN::L,IN::L,IN::T,\

IN::F,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,\

IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::T,IN::L,IN::L,IN::L,IN::L,IN::F,\

IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,\

IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,\

IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,\

IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,\

IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,\

IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,\

IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,\

IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F,IN::F, \

}

Листинг 1 – Таблица входных символов

**ПРИЛОЖЕНИЕ В**

Позиция № строки № столбца Лексема Индекс таблицыid

0 2 1 d -1

1 2 9 t -1

2 2 15 f -1

3 2 20 i 0

4 2 33 ( -1

5 2 34 t -1

6 2 40 i 1

7 2 41 ) -1

8 2 42 { -1

9 3 1 d -1

10 3 9 t -1

11 3 15 i 2

12 3 21 ; -1

13 4 1 i 2

14 4 8 v -1

15 4 10 l 3

16 4 11 ; -1

17 5 2 u -1

18 5 7 ( -1

19 5 8 i 1

20 5 9 b -1

21 5 10 l 3

22 5 11 ) -1

23 6 2 [ -1

24 7 2 i 2

25 7 9 v -1

26 7 11 i 2

27 7 17 v -1

28 7 18 i 1

29 7 19 ; -1

30 8 2 i 1

31 8 4 v -1

32 8 6 i 1

33 8 8 v -1

34 8 10 l 3

35 8 11 ; -1

36 9 2 ] -1

37 9 3 ; -1

38 10 2 r -1

39 10 9 i 2

40 10 15 ; -1

41 11 2 } -1

42 11 3 ; -1

43 12 1 m -1

44 13 1 { -1

45 14 1 a -1

46 14 5 t -1

47 14 12 f -1

48 14 17 i 4

49 14 24 ( -1

50 14 25 ) -1

51 14 26 ; -1

52 15 1 a -1

53 15 5 t -1

54 15 12 f -1

55 15 17 i 5

56 15 24 ( -1

57 15 25 ) -1

58 15 26 ; -1

59 16 1 d -1

60 16 9 t -1

61 16 16 i 6

62 16 20 ; -1

63 17 1 d -1

64 17 9 t -1

65 17 16 i 7

66 17 20 ; -1

67 18 1 i 6

68 18 6 v -1

69 18 8 i 4

70 18 15 ( -1

71 18 16 ) -1

72 18 17 ; -1

73 19 1 i 7

74 19 6 v -1

75 19 8 i 5

76 19 15 ( -1

77 19 16 ) -1

78 19 17 ; -1

79 20 1 w -1

80 20 7 i 6

81 20 11 ; -1

82 21 1 w -1

83 21 7 i 7

84 21 11 ; -1

85 22 1 d -1

86 22 9 t -1

87 22 14 i 8

88 22 16 ; -1

89 23 1 i 8

90 23 4 v -1

91 23 8 l 9

92 23 10 v -1

93 23 14 l 10

94 23 15 ; -1

95 24 1 w -1

96 24 25 l 11

97 24 26 ; -1

98 25 1 w -1

99 25 7 i 8

100 25 9 ; -1

101 26 1 i 8

102 26 4 v -1

103 26 8 l 12

104 26 10 v -1

105 26 14 l 13

106 26 15 ; -1

107 27 1 w -1

108 27 30 l 14

109 27 31 ; -1

110 28 1 w -1

111 28 7 i 8

112 28 9 ; -1

113 29 1 d -1

114 29 9 t -1

115 29 15 i 15

116 29 21 ; -1

117 30 1 i 15

118 30 8 v -1

119 30 10 i 0

120 30 23 ( -1

121 30 24 l 16

122 30 25 ) -1

123 30 27 v -1

124 30 29 i 0

125 30 42 ( -1

126 30 43 l 17

127 30 44 ) -1

128 30 45 ; -1

129 31 1 w -1

130 31 16 l 18

131 31 17 ; -1

132 32 1 w -1

133 32 7 i 15

134 32 13 ; -1

135 33 1 d -1

136 33 9 t -1

137 33 15 i 19

138 33 16 ; -1

139 34 1 i 19

140 34 3 v -1

141 34 5 l 20

142 34 6 ; -1

143 35 1 i 15

144 35 8 v -1

145 35 10 n -1

146 35 11 ( -1

147 35 12 l 21

148 35 16 v -1

149 35 18 l 22

150 35 20 ) -1

151 35 22 v -1

152 35 24 ( -1

153 35 25 l 23

154 35 29 v -1

155 35 31 l 24

156 35 32 ) -1

157 35 34 v -1

158 35 36 l 20

159 35 37 ; -1

160 36 1 w -1

161 36 39 l 25

162 36 40 ; -1

163 37 1 w -1

164 37 7 i 15

165 37 13 ; -1

166 38 1 d -1

167 38 9 t -1

168 38 14 i 26

169 38 16 ; -1

170 39 1 o -1

171 39 3 ( -1

172 39 4 i 26

173 39 6 ) -1

174 40 1 [ -1

175 41 1 w -1

176 41 37 l 27

177 41 38 ; -1

178 42 1 ] -1

179 42 2 ; -1

180 43 1 i 26

181 43 4 v -1

182 43 6 l 28

183 43 10 ; -1

184 44 1 o -1

185 44 3 ( -1

186 44 4 i 26

187 44 6 ) -1

188 45 1 [ -1

189 46 1 w -1

190 46 33 l 29

191 46 34 ; -1

192 47 1 ] -1

193 47 2 ; -1

194 48 1 r -1

195 48 8 l 30

196 48 9 ; -1

197 49 1 } -1

Листинг 1 – Таблица лексем

FST l\_byte(

str,

5,

NODE(1, RELATION('b', 1)),

NODE(1, RELATION('y', 2)),

NODE(1, RELATION('t', 3)),

NODE(1, RELATION('e', 4)),

NODE()

);

FST l\_until(

str,

7,

NODE(1, RELATION('w', 1)),

NODE(1, RELATION('h', 2)),

NODE(1, RELATION('i', 3)),

NODE(1, RELATION('l', 4)),

NODE(1, RELATION('s', 5)),

NODE(1, RELATION('t', 6)),

NODE()

);

FST l\_if(

str,

3,

NODE(1, RELATION('i', 1)),

NODE(1, RELATION('f', 2)),

NODE()

);

FST l\_add(

str,

4,

NODE(1, RELATION('a', 1)),

NODE(1, RELATION('d', 2)),

NODE(1, RELATION('d', 3)),

NODE()

);

FST l\_str(

str,

5,

NODE(1, RELATION('s', 1)),

NODE(1, RELATION('t', 2)),

NODE(1, RELATION('r', 3)),

NODE(1, RELATION('g', 4)),

NODE()

);

FST l\_bool(

str,

5,

NODE(1, RELATION('b', 1)),

NODE(1, RELATION('o', 2)),

NODE(1, RELATION('o', 3)),

NODE(1, RELATION('l', 4)),

NODE()

);

FST l\_true(

str,

5,

NODE(1, RELATION('t', 1)),

NODE(1, RELATION('r', 2)),

NODE(1, RELATION('u', 3)),

NODE(1, RELATION('e', 4)),

NODE()

);

FST l\_false(

str,

6,

NODE(1, RELATION('f', 1)),

NODE(1, RELATION('a', 2)),

NODE(1, RELATION('l', 3)),

NODE(1, RELATION('s', 4)),

NODE(1, RELATION('e', 5)),

NODE()

);

FST l\_numberLiteral(

str,

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 l\_itendificator(

str,

3,

NODE(106,

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),

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('\_', 2)

),

NODE(126,

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),

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('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('\_', 2),

RELATION('0', 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)

),

NODE()

);

FST l\_string(

str,

5,

NODE(1, RELATION('c', 1)),

NODE(1, RELATION('h', 2)),

NODE(1, RELATION('a', 3)),

NODE(1, RELATION('r', 4)),

NODE()

);

FST l\_function(

str,

5,

NODE(1, RELATION('f', 1)),

NODE(1, RELATION('n', 2)),

NODE(1, RELATION('c', 3)),

NODE(1, RELATION('t', 4)),

NODE()

);

FST l\_declare(

str,

9,

NODE(1, RELATION('a', 1)),

NODE(1, RELATION('n', 2)),

NODE(1, RELATION('n', 3)),

NODE(1, RELATION('o', 4)),

NODE(1, RELATION('u', 5)),

NODE(1, RELATION('n', 6)),

NODE(1, RELATION('c', 7)),

NODE(1, RELATION('e', 8)),

NODE()

);

FST l\_return(

str,

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 l\_write(

str,

10,

NODE(1, RELATION('w', 1)),

NODE(1, RELATION('r', 2)),

NODE(1, RELATION('i', 3)),

NODE(1, RELATION('t', 4)),

NODE(2, RELATION('e', 5), RELATION('e', 9)),

NODE(1, RELATION('l', 6)),

NODE(1, RELATION('i', 7)),

NODE(1, RELATION('n', 8)),

NODE(1, RELATION('e', 9)),

NODE()

);

FST l\_main(

str,

5,

NODE(1, RELATION('m', 1)),

NODE(1, RELATION('a', 2)),

NODE(1, RELATION('i', 3)),

NODE(1, RELATION('n', 4)),

NODE()

);

Листинг 2 – Таблица идентификаторов

#pragma once

#define ID\_MAXSIZE 26

#define ID\_CURRENT\_MAXSIZE 10

#define TI\_MAXSIZE 4096

#define TI\_INT\_DEFAULT 0x00000000

#define TI\_STR\_DEFAULT 0x00

#define TI\_NULLIDX 0xffffffff

#define TI\_STR\_MAXSIZE 255

#define FALSYNUMBER 17

namespace IT

{

enum IDDATATYPE { BYTE = 1, STR = 2, CHR = 3, BOOL = 4 };

enum IDTYPE { V = 1, F = 2, P = 3, L = 4 };

struct Entry

{

int idxfirstLE;

char id[ID\_MAXSIZE];

bool isExternal;

IDDATATYPE iddatatype;

IDTYPE idtype;

union {

int vbyte;

char vchar;

bool vbool;

struct

{

char len;

char\* str;

} vstr;

} value;

Entry(int idxfirstLE, const char\* id, IDDATATYPE iddatatype, IDTYPE idtype, bool e = false) {

this->isExternal = e;

this->idxfirstLE = idxfirstLE;

this->iddatatype = iddatatype;

this->idtype = idtype;

if (iddatatype == CHR)

{

this->value.vchar = '\0';

}

if (iddatatype == BYTE)

{

this->value.vbyte = 0;

}

if (iddatatype == STR)

{

this->value.vstr.len = 0;

this->value.vstr.str = nullptr;

}

if (iddatatype == BOOL)

{

this->value.vbool = false;

}

int len = 0;

for (int i = 0; id[i] != '\0' && i < ID\_MAXSIZE; i++)

{

len++;

this->id[i] = id[i];

}

if (len >= ID\_MAXSIZE)

this->id[ID\_MAXSIZE - 1] = '\0';

else

{

this->id[len] = '\0';

}

};

Entry(int idxfirstLE, const char\* id, IDDATATYPE iddatatype, IDTYPE idtype, unsigned int data, bool e = false) {

this->isExternal = e;

this->idxfirstLE = idxfirstLE;

this->iddatatype = iddatatype;

this->idtype = idtype;

this->value.vbyte = data;

int len = 0;

for (int i = 0; id[i] != '\0' && i < ID\_MAXSIZE; i++)

{

len++;

this->id[i] = id[i];

}

if (len >= ID\_MAXSIZE)

this->id[ID\_MAXSIZE - 1] = '\0';

else

{

this->id[len] = '\0';

}

};

Entry(int idxfirstLE, const char\* id, IDDATATYPE iddatatype, IDTYPE idtype, char data, bool e = false) {

this->isExternal = e;

this->idxfirstLE = idxfirstLE;

this->iddatatype = iddatatype;

this->idtype = idtype;

int len = 0;

for (int i = 0; id[i] != '\0' && i < ID\_MAXSIZE; i++)

{

len++;

this->id[i] = id[i];

}

if (len >= ID\_MAXSIZE)

this->id[ID\_MAXSIZE - 1] = '\0';

else

{

this->id[len] = '\0';

}

this->value.vchar = data;

};

Entry(int idxfirstLE, const char\* id, IDDATATYPE iddatatype, IDTYPE idtype, short data, bool e = false) {

this->isExternal = e;

this->idxfirstLE = idxfirstLE;

this->iddatatype = iddatatype;

this->idtype = idtype;

int len = 0;

for (int i = 0; id[i] != '\0' && i < ID\_MAXSIZE; i++)

{

len++;

this->id[i] = id[i];

}

if (len >= ID\_MAXSIZE)

this->id[ID\_MAXSIZE - 1] = '\0';

else

{

this->id[len] = '\0';

}

this->value.vbool = data;

};

Entry(int idxfirstLE, const char\* id, IDDATATYPE iddatatype, IDTYPE idtype, char\* data, bool e = false) {

this->isExternal = e;

this->idxfirstLE = idxfirstLE;

this->iddatatype = iddatatype;

this->idtype = idtype;

int len = 0;

for (int i = 0; id[i] != '\0' && i < ID\_MAXSIZE; i++)

{

len++;

this->id[i] = id[i];

}

if (len >= ID\_MAXSIZE)

this->id[ID\_MAXSIZE - 1] = '\0';

else

{

this->id[len] = '\0';

}

len = 0;

for (int i = 1; data[i] != '\"'; i++)

{

len++;

}

this->value.vstr.len = len + 3;

this->value.vstr.str = new char[len + 3];

for (int i = 0; i < len + 3; i++)

{

this->value.vstr.str[i] = data[i];

}

this->value.vstr.str[len + 3 - 1] = '\0';

this->value.vstr.len = len + 2;

};

Entry() = default;

};

struct IdTable

{

int maxsize;

int size;

Entry\* table;

};

IdTable Create(

int size

);

void Add(

IdTable& idtable,

Entry entry

);

Entry GetEntry(

IdTable& idtable,

int n

);

int IsId(

IdTable& idtable,

char id[ID\_MAXSIZE]

);

int IsLX(

IdTable& idtable,

IDDATATYPE type,

unsigned int data

);

int IsLX(

IdTable& idtable,

IDDATATYPE type,

short data

);

int IsLX(

IdTable& idtable,

IDDATATYPE type,

char data

);

int IsLX(

IdTable& idtable,

IDDATATYPE type,

char\* data

);

bool isUniq(IdTable& idtable,

char id[ID\_MAXSIZE]);

void Delete(IdTable& idtable);

}

Листинг 3 – Конечные автоматы соответствующие ключевым словам

#pragma once

#define LEXEMA\_FIXSIZE 1

#define LT\_MAXSIZE 4096

#define LT\_TI\_NULLIDX 0XFFFFFFFF

#define LEX\_UBYTE 't'

#define LEX\_STRING 't'

#define LEX\_CHAR 't'

#define LEX\_BOOL 't'

#define LEX\_TYPE LEX\_STRING

#define LEX\_ID 'i'

#define LEX\_UNTIL 'u'

#define LEX\_IF 'o'

#define LEX\_MAIN 'm'

#define LEX\_LITERAL 'l'

#define LEX\_FUNCTION 'f'

#define LEX\_DECLARE 'd'

#define LEX\_RETURN 'r'

#define LEX\_ADD 'a'

#define LEX\_WRITE 'w'

#define LEX\_SEMICOLON ';'

#define LEX\_COMMA ','

#define LEX\_LEFTBRACE '{'

#define LEX\_RIGHTBRACE '}'

#define LEX\_LEFT\_SQUAREBRACE '['

#define LEX\_RIGHT\_SQUAREBRACE ']'

#define LEX\_LEFTHESIS '('

#define LEX\_RIGHTHESIS ')'

#define LEX\_PLUS 'v'

#define LEX\_MINUS 'v'

#define LEX\_STAR 'v'

#define LEX\_DIRSLASH 'v'

#define LEX\_EQUAL 'v'

#define LEX\_OPERATOR 'v'

#define LEX\_OR 'v'

#define LEX\_AND 'v'

#define LEX\_INVERSION 'n'

#define LEX\_BOOL\_OPERATOR 'b'

#define EMPTY\_DATA '\0'

namespace LT

{

struct Entry

{

char lexema;

int sn;

int cn;

int idxTI;

char data;

Entry(

char lexema,

int sn,

int cn,

int idxTI,

char symbol = EMPTY\_DATA

) {

this->lexema = lexema;

this->sn = sn;

this->cn = cn;

this->idxTI = idxTI;

this->data = data;

};

Entry() = default;

};

struct LexTable

{

int maxsize;

int size;

Entry\* table;

};

LexTable Create(int size);

void Add(LexTable& lextable,Entry entry);

Entry GetEntry(LexTable& lextable,int n);

void Delete(LexTable& lextable);

}

Листинг 4 – Структура таблицы лексем

# **ПРИЛОЖЕНИЕ Г**

Rule(NS('S'), GRB\_ERROR\_SERIES + 0,

7,

Rule::Chain(7, TS('m'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}')),

Rule::Chain(6, TS('m'), TS('{'), TS('r'), NS('E'), TS(';'), TS('}')),

Rule::Chain(15, TS('d'), TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';'), NS('S')),

Rule::Chain(14, TS('d'), TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';'), NS('S')),

Rule::Chain(14, TS('d'), TS('t'), TS('f'), TS('i'), TS('('), TS(')'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';'), NS('S')),

Rule::Chain(13, TS('d'), TS('t'), TS('f'), TS('i'), TS('('), TS(')'), TS('{'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';'), NS('S')),

Rule::Chain(6, TS('m'), TS('{'), TS('r'), NS('E'), TS(';'), TS('}'))

),

Rule(NS('N'), GRB\_ERROR\_SERIES + 1,

17,

Rule::Chain(4, TS('d'), TS('t'), TS('i'), TS(';')),

Rule::Chain(6, TS('i'), TS('('), NS('F'), TS(')'), TS(';'), NS('N')),

Rule::Chain(5, TS('d'), TS('t'), TS('i'), TS(';'), NS('N')),

Rule::Chain(4, TS('i'), TS('v'), NS('E'), TS(';')),

Rule::Chain(5, TS('i'), TS('v'), NS('E'), TS(';'), NS('N')),

Rule::Chain(9, TS('a'), TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS(';'), NS('N')),

Rule::Chain(8, TS('a'), TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS(';')),

Rule::Chain(8, TS('a'), TS('t'), TS('f'), TS('i'), TS('('), TS(')'), TS(';'), NS('N')),

Rule::Chain(7, TS('a'), TS('t'), TS('f'), TS('i'), TS('('), TS(')'), TS(';')),

Rule::Chain(4, TS('w'), TS('i'), TS(';'), NS('N')),

Rule::Chain(3, TS('w'), TS('i'), TS(';')),

Rule::Chain(4, TS('w'), TS('l'), TS(';'), NS('N')),

Rule::Chain(3, TS('w'), TS('l'), TS(';')),

Rule::Chain(8, TS('o'), TS('('), NS('B'), TS(')'), TS('['), NS('N'), TS(']'), TS(';')),

Rule::Chain(9, TS('o'), TS('('), NS('B'), TS(')'), TS('['), NS('N'), TS(']'), TS(';'), NS('N')),

Rule::Chain(8, TS('u'), TS('('), NS('B'), TS(')'), TS('['), NS('N'), TS(']'), TS(';')),

Rule::Chain(9, TS('u'), TS('('), NS('B'), TS(')'), TS('['), NS('N'), TS(']'), TS(';'), NS('N'))

),

Rule(NS('E'), GRB\_ERROR\_SERIES + 2,

16,

Rule::Chain(1, TS('i')),

Rule::Chain(1, TS('l')),

Rule::Chain(2, TS('n'), TS('i')),

Rule::Chain(2, TS('n'), TS('l')),

Rule::Chain(3, TS('('), NS('E'), TS(')')),

Rule::Chain(4, TS('i'), TS('('), NS('W'), TS(')')),

Rule::Chain(3, TS('i'), TS('('), TS(')')),

Rule::Chain(2, TS('i'), NS('M')),

Rule::Chain(2, TS('l'), NS('M')),

Rule::Chain(3, TS('n'), TS('i'), NS('M')),

Rule::Chain(3, TS('n'), TS('l'), NS('M')),

Rule::Chain(4, TS('n'), TS('('), NS('E'), TS(')')),

Rule::Chain(5, TS('n'), TS('('), NS('E'), TS(')'), NS('M')),

Rule::Chain(4, TS('('), NS('E'), TS(')'), NS('M')),

Rule::Chain(4, TS('('), NS('E'), TS(')'), NS('M')),

Rule::Chain(5, TS('i'), TS('('), NS('W'), TS(')'), NS('M'))

),

Rule(NS('M'), GRB\_ERROR\_SERIES + 3,

4,

Rule::Chain(2, TS('v'), NS('E')),

Rule::Chain(4, TS('v'), TS('('), NS('E'), TS(')')),

Rule::Chain(5, TS('v'), TS('('), NS('E'), TS(')'), NS('M')),

Rule::Chain(3, TS('v'), NS('E'), NS('M'))

),

Rule(NS('F'), GRB\_ERROR\_SERIES + 4,

5,

Rule::Chain(2, TS('t'), TS('i')),

Rule::Chain(4, TS('t'), TS('i'), TS(','), NS('F')),

Rule::Chain(1, TS('i')),

Rule::Chain(3, TS('i'), TS(','), NS('F')),

Rule::Chain(3, TS('l'), TS(','), NS('F'))

),

Rule(NS('W'), GRB\_ERROR\_SERIES + 5,

4,

Rule::Chain(1, TS('i')),

Rule::Chain(1, TS('l')),

Rule::Chain(3, TS('i'), TS(','), NS('W')),

Rule::Chain(3, TS('l'), TS(','), NS('W'))

),

Rule(NS('B'), GRB\_ERROR\_SERIES + 6,

6,

Rule::Chain(3, TS('i'), TS('b'), TS('i')),

Rule::Chain(3, TS('i'), TS('b'), TS('l')),

Rule::Chain(3, TS('l'), TS('b'), TS('i')),

Rule::Chain(3, TS('l'), TS('b'), TS('l')),

Rule::Chain(1, TS('i')),

Rule::Chain(1, TS('l'))

)

Листинг 1 – Грамматика языка ZEO-2024

struct MfstDiagnosis

{

short lenta\_position;

RC\_STEP rc\_step;

short nrule;

short nrule\_chain;

MfstDiagnosis();

MfstDiagnosis(

short plenta\_position,

RC\_STEP prc\_step,

short pnrule,

short pnrule\_chain

);

} diagnosis[MFST\_DIAGN\_NUMBER];

GRBALPHABET\* lenta;

short lenta\_position;

short nrule;

short nrulechain;

short lenta\_size;

GRB::Greibach greibach;

LEX::LEX lex;

MFSTSTACK st;

bool shallWrite;

use\_container<std::stack<MfstState>> storestate;

Mfst();

Mfst(

LEX::LEX plex,

GRB::Greibach pgreibach,

bool shouldWrite

);

char\* getCSt(char\* buf);

char\* getCLenta(char\* buf, short pos, short n = 25);

char\* getDiagnosis(short n, char\* buf);

bool savestate(const Log::LOG& log);

bool reststate(const Log::LOG& log);

bool push\_chain(

GRB::Rule::Chain chain

);

RC\_STEP step(const Log::LOG& log);

bool start(const Log::LOG& log);

bool savediagnosis(

RC\_STEP pprc\_step

);

Листинг 2 – структура конечного магазинного автомата

# **ПРИЛОЖЕНИЕ Д**

Шаг :Правило Входная лента Стек

0 :S->dtfi(F){NrE;};S dtfi(ti){u(ibl)[wl;ivivl; S$

1 : SAVESTATE: 1

1 : dtfi(ti){u(ibl)[wl;ivivl; dtfi(F){NrE;};S$

2 : tfi(ti){u(ibl)[wl;ivivl;] tfi(F){NrE;};S$

3 : fi(ti){u(ibl)[wl;ivivl;]; fi(F){NrE;};S$

4 : i(ti){u(ibl)[wl;ivivl;];r i(F){NrE;};S$

5 : (ti){u(ibl)[wl;ivivl;];rl (F){NrE;};S$

6 : ti){u(ibl)[wl;ivivl;];rl; F){NrE;};S$

7 :F->ti ti){u(ibl)[wl;ivivl;];rl; F){NrE;};S$

8 : SAVESTATE: 2

8 : ti){u(ibl)[wl;ivivl;];rl; ti){NrE;};S$

9 : i){u(ibl)[wl;ivivl;];rl;} i){NrE;};S$

10 : ){u(ibl)[wl;ivivl;];rl;}; ){NrE;};S$

11 : {u(ibl)[wl;ivivl;];rl;};m {NrE;};S$

12 : u(ibl)[wl;ivivl;];rl;};m{ NrE;};S$

13 :N->u(B)[N]; u(ibl)[wl;ivivl;];rl;};m{ NrE;};S$

14 : SAVESTATE: 3

14 : u(ibl)[wl;ivivl;];rl;};m{ u(B)[N];rE;};S$

15 : (ibl)[wl;ivivl;];rl;};m{d (B)[N];rE;};S$

16 : ibl)[wl;ivivl;];rl;};m{dt B)[N];rE;};S$

17 :B->ibi ibl)[wl;ivivl;];rl;};m{dt B)[N];rE;};S$

18 : SAVESTATE: 4

18 : ibl)[wl;ivivl;];rl;};m{dt ibi)[N];rE;};S$

19 : bl)[wl;ivivl;];rl;};m{dti bi)[N];rE;};S$

20 : l)[wl;ivivl;];rl;};m{dti; i)[N];rE;};S$

21 : TS\_NOK/NS\_NORULECHAIN

21 : RESSTATE

Листинг 1 – Начало синтаксического анализатора

1132:N->wi; wl;];rl;} N];rE;}$

1133: SAVESTATE: 83

1133: wl;];rl;} wi;];rE;}$

1134: l;];rl;} i;];rE;}$

1135: TS\_NOK/NS\_NORULECHAIN

1135: RESSTATE

1135: wl;];rl;} N];rE;}$

1136:N->wl;N wl;];rl;} N];rE;}$

1137: SAVESTATE: 83

1137: wl;];rl;} wl;N];rE;}$

1138: l;];rl;} l;N];rE;}$

1139: ;];rl;} ;N];rE;}$

1140: ];rl;} N];rE;}$

1141: TNS\_NORULECHAIN/NS\_NORULE

1141: RESSTATE

1141: wl;];rl;} N];rE;}$

1142:N->wl; wl;];rl;} N];rE;}$

1143: SAVESTATE: 83

1143: wl;];rl;} wl;];rE;}$

1144: l;];rl;} l;];rE;}$

1145: ;];rl;} ;];rE;}$

1146: ];rl;} ];rE;}$

1147: ;rl;} ;rE;}$

1148: rl;} rE;}$

1149: l;} E;}$

1150:E->l l;} E;}$

1151: SAVESTATE: 84

1151: l;} l;}$

1152: ;} ;}$

1153: } }$

1154: $

1155: LENTA\_END

1156: ------>LENTA\_END

Листинг 2 - Конец синтаксического анализа

0 :всего строк 238, синтаксический анализ выполнен без ошибок

0 : S->dtfi(F){NrE;};S

5 : F->ti

9 : N->u(B)[N];

11 : B->ibl

16 : N->wl;N

19 : N->ivE;

21 : E->iM

22 : M->vE

23 : E->l

28 : E->l

32 : S->m{NrE;}

34 : N->dti;N

38 : N->ivE;N

40 : E->i(W)

42 : W->l

45 : N->wl;N

48 : N->wi;N

51 : N->atfi(F);N

56 : F->ti,F

59 : F->ti

63 : N->atfi(F);N

68 : F->ti

72 : N->dti;N

76 : N->ivE;N

78 : E->i(W)

80 : W->l,W

82 : W->l

85 : N->wl;N

88 : N->wi;N

91 : N->dti;N

95 : N->ivE;N

97 : E->i(W)

99 : W->l,W

101 : W->l

104 : N->wl;N

107 : N->wi;N

110 : N->dti;N

114 : N->ivE;N

116 : E->lM

117 : M->vE

118 : E->l

120 : N->wl;N

123 : N->wi;N

126 : N->dti;N

130 : N->ivE;N

132 : E->i(W)

134 : W->i

137 : N->wl;N

140 : N->wi;N

143 : N->dti;N

147 : N->ivE;N

149 : E->l

151 : N->wl;N

154 : N->wi;N

157 : N->dti;N

161 : N->ivE;N

163 : E->l

165 : N->wl;N

168 : N->wi;N

171 : N->dti;N

175 : N->ivE;N

177 : E->lM

178 : M->vE

179 : E->l

181 : N->wl;N

184 : N->wi;N

187 : N->dti;N

191 : N->ivE;N

193 : E->lM

194 : M->vE

195 : E->l

197 : N->wl;N

200 : N->wi;N

203 : N->dti;N

207 : N->o(B)[N];N

209 : B->i

212 : N->wl;

217 : N->ivE;N

219 : E->l

221 : N->wl;N

224 : N->o(B)[N];

226 : B->i

229 : N->wl;

235 : E->l

Листинг 3 – Пример разбора синтаксического анализатора

# **ПРИЛОЖЕНИЕ Е**

#include <stack>

#include <vector>

#include <iostream>

#include "PolishNotation.h"

#include "Error.h"

namespace PolishNotation {

template <typename T>

struct container : T

{

using T::T;

using T::c;

};

std::string toString(int n) {

char buf[40];

sprintf\_s(buf, "%d", n);

return buf;

}

bool find\_elem(std::stack<char> stack, size\_t size, char elem) {

for (size\_t i = 0; i < size; i++)

if (stack.top() == elem)

return true;

else

stack.pop();

return false;

}

int get\_priority(char a)

{

switch (a)

{

case '(':

return 0;

case ')':

return 0;

case ',':

return 1;

case '-':

return 2;

case '+':

return 2;

case '\*':

return 3;

case '%':

return 3;

case '/':

return 3;

case '\\':

return 3;

case ':':

return 3;

case '&':

return 3;

case '|':

return 3;

case '~':

return 4;

default: {

return 0;

}

}

}

void fixIt(LT::LexTable& lextable, const std::string& str, size\_t length, size\_t pos, const std::vector<int>& ids) {

std::cout << str << std::endl;

for (size\_t i = 0, q = 0; i < str.size(); i++) {

lextable.table[pos + i].lexema = str[i];

if (lextable.table[pos + i].lexema == LEX\_ID || lextable.table[pos + i].lexema == LEX\_LITERAL) {

lextable.table[pos + i].idxTI = ids[q];

q++;

}

else

lextable.table[pos + i].idxTI = LT\_TI\_NULLIDX;

}

int temp = str.size() + pos;

for (size\_t i = 0; i < length - str.size(); i++) {

lextable.table[temp + i].idxTI = LT\_TI\_NULLIDX;

lextable.table[temp + i].lexema = '#';

lextable.table[temp + i].sn = -1;

}

}

bool PolishNotation(int lextable\_pos, LT::LexTable& lextable, IT::IdTable& idtable)

{

container<std::stack<char>> stack;

std::string PolishString;

std::vector<int> ids;

int operators\_count = 0, operands\_count = 0, iterator = 0, right\_counter = 0, left\_counter = 0, params\_counter = 0;

for (int i = lextable\_pos; i < lextable.size; i++, iterator++) {

char lexem = lextable.table[i].lexema;

char data = lextable.table[i].data;

size\_t stack\_size = stack.size();

if (idtable.table[lextable.table[i].idxTI].idtype == IT::IDTYPE::F) {

stack.push('@');

operands\_count--;

}

switch (lexem) {

case LEX\_OPERATOR:

{

if (!stack.empty() && stack.top() != LEX\_LEFTHESIS) {

while (!stack.empty() && get\_priority(data) <= get\_priority(stack.top())) {

PolishString += stack.top();

stack.pop();

}

}

stack.push(data);

operators\_count++;

break;

}

case LEX\_INVERSION:

{

stack.push(data);

break;

}

case LEX\_COMMA:

{

while (!stack.empty()) {

if (stack.top() == LEX\_LEFTHESIS)

break;

PolishString += stack.top();

stack.pop();

}

operands\_count--;

break;

}

case LEX\_LEFTHESIS:

{

left\_counter++;

stack.push(lexem);

break;

}

case LEX\_RIGHTHESIS:

{

right\_counter++;

if (!find\_elem(stack, stack\_size, LEX\_LEFTHESIS))

return false;

while (stack.top() != LEX\_LEFTHESIS) {

PolishString += stack.top();

stack.pop();

}

stack.pop();

if (!stack.empty() && stack.top() == '@') {

PolishString += stack.top() + toString(params\_counter - 1);

params\_counter = 0;

stack.pop();

}

break;

}

case LEX\_SEMICOLON:

{

if (operators\_count != 0 && operands\_count != 0)

if ((!stack.empty() && (stack.top() == LEX\_RIGHTHESIS || stack.top() == LEX\_LEFTHESIS))

|| right\_counter != left\_counter || operands\_count - operators\_count != 1)

return false;

while (!stack.empty()) {

PolishString += stack.top();

stack.pop();

}

fixIt(lextable, PolishString, iterator, lextable\_pos, ids);

return true;

break;

}

case LEX\_ID: {

if (std::find(stack.c.begin(), stack.c.begin(), '@') != stack.c.end())

params\_counter++;

PolishString += lexem;

if (lextable.table[i].idxTI != LT\_TI\_NULLIDX)

ids.push\_back(lextable.table[i].idxTI);

operands\_count++;

if (!stack.empty() && stack.top() == '~')

{

PolishString += stack.top();

stack.pop();

}

break;

}

case LEX\_LITERAL: {

if (std::find(stack.c.begin(), stack.c.begin(), '@') != stack.c.end())

params\_counter++;

PolishString += lexem;

if (lextable.table[i].idxTI != LT\_TI\_NULLIDX)

ids.push\_back(lextable.table[i].idxTI);

operands\_count++;

if (!stack.empty() && stack.top() == '~')

{

PolishString += stack.top();

stack.pop();

}

break;

}

}

}

return true;

}

void DoPolish(LEX::LEX t) {

for (int i = 0; i < t.lextable.size; i++)

if (t.lextable.table[i].lexema == LEX\_EQUAL)

if (!PolishNotation(i + 1, t.lextable, t.idtable))

throw ERROR\_THROW(130);

for (int i = 0; i < t.lextable.size; i++)

if (t.lextable.table[i].lexema == '+' || t.lextable.table[i].lexema == '-' || t.lextable.table[i].lexema == '\*' ||

t.lextable.table[i].lexema == '/' || t.lextable.table[i].lexema == '\\' || t.lextable.table[i].lexema == ':' ||

t.lextable.table[i].lexema == '%' || t.lextable.table[i].lexema == '&' || t.lextable.table[i].lexema == '|'

|| t.lextable.table[i].lexema == '~')

{

t.lextable.table[i].data = t.lextable.table[i].lexema;

t.lextable.table[i].lexema = LEX\_OPERATOR;

}

}

}

Листинг 1 - алгоритм преобразования выражений к обратной польской записи, реализованный на C++

# **ПРИЛОЖЕНИЕ Ж**

.586

.model flat, stdcall

includelib libucrt.lib

includelib kernel32.lib

includelib ../Debug/ZEO-2024Lib.lib

ExitProcess PROTO :DWORD

compare PROTO :DWORD, :DWORD

absolute PROTO :BYTE

outputbyte PROTO :BYTE

outputlinebyte PROTO :BYTE

outputchar PROTO :BYTE

outputlinechar PROTO :BYTE

outputstr PROTO :DWORD

outputlinestr PROTO :DWORD

outputbool PROTO :BYTE

outputlinebool PROTO :BYTE

.stack 4096

.const

divideOnZeroExeption BYTE "Attemption of dividing by 0.", 0 ;STR, для вывода ошибки при делении на ноль

FindFactor\_$LEX1 BYTE 1 ;UBYTE

main$LEX4 BYTE "str", 0 ;STR

main$LEX5 BYTE "str1", 0 ;STR

main$LEX6 BYTE "Результат выполнения функции сравнения 2 строк:", 0 ;STR

main$LEX8 BYTE 10 ;UBYTE

main$LEX9 BYTE 'Z' ;CHR

main$LEX10 BYTE ' ' ;CHR

main$LEX11 BYTE "'Z' - ' '(space)", 0 ;STR

main$LEX12 BYTE 'A' ;CHR

main$LEX13 BYTE ' ' ;CHR

main$LEX14 BYTE "'A' + ' '(tabulation)", 0 ;STR

main$LEX15 BYTE 5 ;UBYTE

main$LEX17 BYTE "!5 + !4", 0 ;STR

main$LEX18 BYTE 3 ;UBYTE

main$LEX19 BYTE 240 ;UBYTE

main$LEX20 BYTE 15 ;UBYTE

main$LEX21 BYTE 127 ;UBYTE

main$LEX22 BYTE 8 ;UBYTE

main$LEX25 BYTE "~(240 | 15) + (127 & 8) : 3 = ", 0 ;STR

main$LEX26 BYTE "condition that will not work", 0 ;STR

main$LEX27 BYTE 1 ;BOOL

main$LEX28 BYTE "condition that will work", 0 ;STR

main$LEX29 BYTE 0 ;UBYTE

.data

FindFactor\_answer BYTE 0 ;UBYTE

mainstr DWORD 0 ;STR

maincomparemen BYTE 0 ;UBYTE

mainnum BYTE 0 ;UBYTE

mainch BYTE 0 ;CHR

mainresult BYTE 0 ;UBYTE

maina BYTE 0 ;UBYTE

mainbc BYTE 0 ;BOOL

.code

$FindFactor PROC uses ebx ecx edi esi , FindFactor\_a: BYTE

; String #4 :ivl

movzx eax, FindFactor\_$LEX1

push eax

pop eax

mov FindFactor\_answer, al

While17Start:

movzx eax, FindFactor\_a

movzx ebx, FindFactor\_$LEX1

cmp eax, ebx

jl While17End

; String #7 :iviiv

movzx eax, FindFactor\_answer

push eax

movzx eax, FindFactor\_a

push eax

pop ebx

pop eax

mul ebx

push eax

pop eax

mov FindFactor\_answer, al

; String #8 :ivilv

movzx eax, FindFactor\_a

push eax

movzx eax, FindFactor\_$LEX1

push eax

pop ebx

pop eax

sub eax, ebx

push eax

pop eax

mov FindFactor\_a, al

jmp While17Start

While17End:

movzx eax, FindFactor\_answer

ret

$FindFactor ENDP

main PROC

; String #20 :ivill@2#

invoke compare, offset main$LEX4, offset main$LEX5

push eax ;результат функции

pop eax

mov maincomparemen, al

push offset main$LEX6

CALL outputlinestr

push eax

movzx eax, maincomparemen

push eax

CALL outputlinebyte

pop eax

; String #24 :ivllv

movzx eax, FindFactor\_$LEX1

push eax

movzx eax, main$LEX8

push eax

pop ebx

pop eax

sub eax, ebx

push eax

pop eax

mov mainnum, al

push eax

movzx eax, mainnum

push eax

CALL outputlinebyte

pop eax

; String #26 :ivii@1

invoke absolute, mainnum

push eax ;результат функции

pop eax

mov mainnum, al

push eax

movzx eax, mainnum

push eax

CALL outputlinebyte

pop eax

; String #29 :ivllv

movzx eax, main$LEX9

push eax

movzx eax, main$LEX10

push eax

pop ebx

pop eax

sub eax, ebx

push eax

pop eax

mov mainch, al

push offset main$LEX11

CALL outputstr

; String #31 :ivllv

movzx eax, main$LEX12

push eax

movzx eax, main$LEX13

push eax

pop ebx

pop eax

add eax, ebx

push eax

pop eax

mov mainch, al

push offset main$LEX14

CALL outputlinestr

push eax

movzx eax, mainch

push eax

CALL outputlinechar

pop eax

; String #35 :ivil@1il@1v

invoke $FindFactor, main$LEX15

push eax ;результат функции

invoke $FindFactor, main$LEX15

push eax ;результат функции

pop ebx

pop eax

add eax, ebx

push eax

pop eax

mov mainresult, al

push offset main$LEX17

CALL outputlinestr

push eax

movzx eax, mainresult

push eax

CALL outputlinebyte

pop eax

; String #39 :ivl

movzx eax, main$LEX18

push eax

pop eax

mov maina, al

; String #40 :ivllvvllvllvvv######

movzx eax, main$LEX19

push eax

movzx eax, main$LEX20

push eax

pop ebx

pop eax

or eax, ebx

push eax

pop eax

not eax

push eax

movzx eax, main$LEX21

push eax

movzx eax, main$LEX22

push eax

pop ebx

pop eax

and eax, ebx

push eax

movzx eax, FindFactor\_$LEX1

push eax

movzx eax, FindFactor\_$LEX1

push eax

pop ebx

pop eax

sub eax, ebx

push eax

pop ebx

pop eax

push edx ; сохраняем данные регистра edx

mov edx, 0

TEST EBX, EBX

JZ div\_by\_0

div ebx

pop edx

push eax

pop ebx

pop eax

add eax, ebx

push eax

pop eax

mov mainresult, al

push offset main$LEX25

CALL outputlinestr

push eax

movzx eax, mainresult

push eax

CALL outputlinebyte

pop eax

If198Start:

movzx eax, mainbc

mov ebx, 1

cmp eax, ebx

jnz If198End

push offset main$LEX26

CALL outputlinestr

If198End:

; String #48 :ivl

movzx eax, main$LEX27

push eax

pop eax

mov mainbc, al

If212Start:

movzx eax, mainbc

mov ebx, 1

cmp eax, ebx

jnz If212End

push offset main$LEX28

CALL outputlinestr

If212End:

movzx eax, main$LEX29

jmp endPoint

div\_by\_0:

push offset divideOnZeroExeption

CALL outputstr

endPoint:

invoke ExitProcess, eax

main ENDP

end main