МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет ФИТ Кафедра ПИ как Специальность 1-40 01 01 Программное обеспечение информационных технологий

УТВЕРЖДАЮ

Заведующий кафедрой

\_\_\_\_\_\_\_\_\_\_\_\_

Н.В. Пацей

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2022 г.

ЗАДАНИЕ

к курсовому проектированию по дисциплине

«Конструирование программного обеспечения»

Специальность: ПОИТ Группа: 4 в

Студент: Точило Олег Вячеславович а

1.Тема проекта Разработка компилятора TOV-2022 в

2. Срок сдачи студентом проекта: \_\_\_ декабря 202\_ г.

3. Исходные данные к проекту: Разработка программы осуществляется на языке C++ (стандартизации International Standard ISO/IEC 14882:2020(E) Programming Language C++20) в среде разработки Visual Studio 2022. Типы данных: целочисленный, строковый. Операции языка: сравнения (равенство, неравенство, меньше, больше, меньше или равно, больше или равно). Литералы: целые в восьмеричном и двоичном представлении, строковые или символьные в произвольном представлении. Функции стандартной библиотеки: лексикографическое сравнение строк, преобразование строки в число, поиск подстроки в строке. Инструкции: условные операторы.

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

* Введение
* Спецификация языка программирования
* Структура транслятора
* Разработка лексического анализатора
* Разработка синтаксического анализатора
* Разработка семантического анализатора
* Вычисление выражений
* Генерация кода
* Тестирование транслятора (и/или Разработка и тестирование
* интерпретатора)
* Заключение
* Литература
* Приложения

1. 5. Перечень графического материала (с точным указанием обязательных чертежей)
2. Граф дерева разбора

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

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

7. Календарный план

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Наименование этапов курсового проекта | Срок выполнения этапов проекта | Примечание |
| 1 | Спецификация специализированного языка TOV-2022 |  |  |
| 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 15.09.2022

Руководитель Наркевич Аделина Сергеевна ы

(подпись, фамилия, имя, отчество)

Задание принял к исполнению а

(подпись, фамилия, имя, отчество)

**Содержание**

[Глава 1 Спецификация языка программирования 5](#_Toc122284038)

[1.1 Характеристика языка программирования 5](#_Toc122284039)

[1.2 Определение алфавита языка программирования 5](#_Toc122284040)

[1.3 Применяемые сепараторы 5](#_Toc122284041)

[1.4 Применяемые кодировки 5](#_Toc122284042)

[1.5 Типы данных 6](#_Toc122284043)

[1.6 Преобразование типов 6](#_Toc122284044)

[1.7 Идентификаторы 7](#_Toc122284045)

[1.8 Литералы 7](#_Toc122284046)

[1.9 Объявление данных 7](#_Toc122284047)

[1.10 Инициализация данных 7](#_Toc122284048)

[1.11 Инструкции языка 7](#_Toc122284049)

[1.12 Операции языка 8](#_Toc122284050)

[1.13 Выражения и их вычисление 8](#_Toc122284051)

[1.14 Конструкции языка 8](#_Toc122284052)

[1.15 Область видимости идентификаторов 9](#_Toc122284053)

[1.16 Семантические проверки 9](#_Toc122284054)

[1.17 Распределение оперативной памяти на этапе выполнения 9](#_Toc122284055)

[1.18 Стандартная библиотека и её состав 9](#_Toc122284056)

[1.19 Ввод и вывод данных 10](#_Toc122284057)

[1.20 Точка входа 10](#_Toc122284058)

[1.21 Препроцессор 10](#_Toc122284059)

[1.22 Соглашения о вызовах 10](#_Toc122284060)

[1.23 Объектный код 10](#_Toc122284061)

[1.24 Классификация сообщений транслятора 10](#_Toc122284062)

[1.25 Контрольный пример 11](#_Toc122284063)

[2.2 Перечень входных параметров транслятора 12](#_Toc122284064)

[2.3 Протоколы, формируемые транслятором 12](#_Toc122284065)

[Глава 3. Разработка лексического анализатора 13](#_Toc122284066)

[3.1 Структура лексического анализатора 13](#_Toc122284067)

[3.2 Контроль входных символов 13](#_Toc122284068)

[3.3 Удаление избыточных символов 13](#_Toc122284069)

[3.4 Перечень ключевых слов 13](#_Toc122284070)

[3.5 Основные структуры данных 15](#_Toc122284071)

[3.6 Структура и перечень сообщений лексического анализатора 16](#_Toc122284072)

[3.7 Принцип обработки ошибок 17](#_Toc122284073)

[3.8 Параметры лексического анализатора 17](#_Toc122284074)

[3.9 Алгоритм лексического анализа 17](#_Toc122284075)

[3.10 Контрольный пример 18](#_Toc122284076)

[Глава 4. Разработка синтаксического анализатора 19](#_Toc122284077)

[4.1 Структура синтаксического анализатора 19](#_Toc122284078)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 19](#_Toc122284079)

[4.3 Построение конечного магазинного автомата 21](#_Toc122284080)

[4.4 Основные структуры данных 21](#_Toc122284081)

[4.5 Описание алгоритма синтаксического разбора 21](#_Toc122284082)

[4.6 Структура и перечень сообщений синтаксического анализатора 22](#_Toc122284083)

[4.7 Параметры синтаксического анализатора и режимы его работы 22](#_Toc122284084)

[4.8 Принцип обработки ошибок 22](#_Toc122284085)

[4.9 Контрольный пример 23](#_Toc122284086)

[Глава 5. Разработка семантического анализатора 24](#_Toc122284087)

[5.1 Структура семантического анализатора 24](#_Toc122284088)

[5.2 Функции семантического анализатора 24](#_Toc122284089)

[5.3 Структура и перечень сообщений семантического анализатора 24](#_Toc122284090)

[5.4 Принцип обработки ошибок 25](#_Toc122284091)

[Глава 6. Преобразование выражений 26](#_Toc122284092)

[6.1 Выражения, допускаемые языком 26](#_Toc122284093)

[6.2 Польская запись и принцип ее построения 26](#_Toc122284094)

[6.3 Программная реализация обработки выражений 27](#_Toc122284095)

[6.4 Контрольный пример 27](#_Toc122284096)

[Глава 7. Генерация кода 28](#_Toc122284097)

[7.1 Структура генератора кода 28](#_Toc122284098)

[7.2 Представление типов данных в оперативной памяти 28](#_Toc122284099)

[7.3 Статическая библиотека 28](#_Toc122284100)

[7.4 Особенности алгоритма генерации кода 29](#_Toc122284101)

[7.5 Контрольный пример 30](#_Toc122284102)

[Глава 8. Тестирование транслятора 31](#_Toc122284103)

[8.1 Тестирование проверки на допустимость символов 31](#_Toc122284104)

[8.2 Тестирование лексического анализатора 31](#_Toc122284105)

[8.3 Тестирование синтаксического анализатора 31](#_Toc122284106)

[8.4 Тестирование семантического анализатора 32](#_Toc122284107)

[Приложение А 33](#_Toc122284108)

[Приложение Б 34](#_Toc122284109)

[Приложение В 35](#_Toc122284110)

[Приложение Г 36](#_Toc122284111)

[Приложение Д 41](#_Toc122284112)

[Приложение Е 43](#_Toc122284113)

[Приложение Ж 44](#_Toc122284114)

[Приложение З 45](#_Toc122284115)

**Введение**

Целью данного курсового проекта является разработка языка программирования TOV-2022 и компилятора для него. Написание компилятора будет осуществляться на языке C++.

Данная пояснительная записка будет включать в себя описания составляющих частей языка программирования и компилятора для него: спецификацию языка TOV-2022, структура транслятора, лексического, синтаксического и семантического анализаторов, принцип вычисления выражений, структура генератора кода и тестирование транслятора.

Для выполнения цели курсового проекта были поставлены следующие задачи:

* создание спецификации языка программирования (пункт 1 пояснительной записки);
* разработка структуры транслятора (пункт 2)
* разработка лексического анализатора (пункт 3)
* разработка синтаксического анализатора (пункт 4
* разработка семантического анализатора
* вычисление выражений
* генерация кода на языке ассемблера
* тестирование транслятора на контрольных примерах.

**Глава 1 Спецификация языка программирования**

## 1.1 Характеристика языка программирования

Язык программирования TOV-2022 – функциональный строго типизированный компилируемый язык.

## 1.2 Определение алфавита языка программирования

Алфавит языка TOV-2022 основывается на таблице Windows-1251, представленной на рисунке 1.1.

Буква –> {a | b | … | z | A | B | … | Z}

Цифра –> {0 | 1 | … | 9}

Специальный символ -> {( | ) | { | } | , | пробел | \t | \n | < | > | = | + | – | \* | /}

Символ языка –> {Буква | Цифра | Специальный символ}

## 1.3 Применяемые сепараторы

Применяемые сепараторы и их назначение представлено в таблице 1.1.

Таблица 1.1 – применяемые сепараторы

|  |  |
| --- | --- |
| {} | определяют границы тела функции и условных блоков |
| () | увеличивают приоритет выражения |
| пробел | разделяет лексемы |
| , | разделяет параметры функции |
| < > >= <= == != | разделяют операнды в условном выражении |
| ; | разделяет инструкции |
| “ | определяет границы строковых литералов |
| ‘ | определяет границы символьных литералов |
| = | разделяет присваиваемое выражение и переменную |
| + – \* / | разделяют операнды в арифметических выражениях |

## 1.4 Применяемые кодировки

Код на языке TOV-2022 хранится в кодировке Windows-1251 – стандартной кодировке русскоязычных версий ОС Windows, представленной на рисунке 1.1.

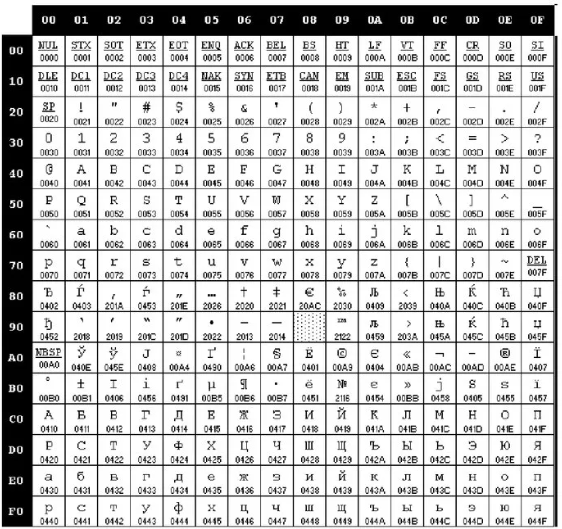


Рисунок 1.1 Таблица кодировки Windows-1251

## 1.5 Типы данных

Типы данных языка TOV-2022 представлены в таблице 1.2.

Таблица 1.2 – типы данных

|  |  |
| --- | --- |
| string | строка однобайтовых символов длиной до 255 символов, хранит адрес первого элемента строки, хранящего длину строки |
| number | целочисленный, 4 байта, по умолчанию инициализируется нулём |
| symbol | символьный, является строкой нулевой или единичной длины, занимает 2 байта |

## 1.6 Преобразование типов

Преобразование типов не предусмотрено.

## 1.7 Идентификаторы

В качестве идентификаторов допускаются любые комбинации прописных символов латинского алфавита ([a; z]) длиной от 1 до 32, не совпадающие с ключевыми словами. В случае, если длина идентификатора превышает 32 символа, выполняется усечение до 32 символов.

## 1.8 Литералы

Целочисленные литералы принимают значения от 0 до 216-1. В случае превышения данного предела значение автоматически уменьшится до 216-1. Целочисленный литерал не может быть отрицательным. Для получения отрицательного значения необходимо отнять большее число от меньшего

Строковые литералы должны начинаться и заканчиваться символом одинарной кавычки. Между кавычками могут находиться любые разрешённые символы, кроме другой двойной кавычки. Символьные литералы отличаются тем, что заключаются в двойные кавычки и содержат либо один, либо ни одного символа между ними.

## 1.9 Объявление данных

Переменные и пользовательские функции должны быть объявлены перед использованием. Для объявления переменной необходимо использовать ключевое слово var и указать идентификатор переменной. Для объявления функции необходимо использовать ключевое слово function и указать идентификатор, список параметров и тело функции.

## 1.10 Инициализация данных

Для присваивания переменной значения переменная должна быть уже объявлена, а новое значение должно стоять справа от знака равенства.

## 1.11 Инструкции языка

Инструкции языка TOV-2022 представлены в таблице 1.3.

Таблица 1.3 – инструкции языка TOV-2022

|  |  |
| --- | --- |
| Инструкция | Запись на языке TOV-2022 |
| Объявление переменной | var <тип данных> <идентификатор>; |
| Возврат из функции | return <выражение>; |
| Вывод данных: | print <выражение>; |
| Вызов подпрограммы | <идентификатор функции> ([<список параметров>]); |
| Присваивание | <идентификатор> = <выражение>; |

Продолжение таблицы 1.3

|  |  |
| --- | --- |
| Инструкция | Запись на языке TOV-2022 |
| Условное выражение | [else]if(<условное выражение>){<условный блок>}|else{<условный блок>} |
| Условный блок | <<вывод данных>|<присваивание>>[<условный блок>] |

## 1.12 Операции языка

В языке TOV-2022 присутствуют логические операции (равенство, неравенство, больше, меньше, больше или равно, меньше или равно), арифметические операции (сложение, вычитание, умножение, деление) и операция присваивания. Арифметические операции, заключённые в круглые скобки, имеют наивысший приоритет. Операции сравнения имеют самый низкий приоритет.

Операторы языка TOV-2022 представлены в таблице 1.1. Приоритет операторов представлен в таблице 1.4.

Таблица 1.4 – приоритет операторов

|  |  |
| --- | --- |
| Операции | Приоритет |
| + – | 3 |
| \* / | 4 |
| < > <= >= == != | 2 |

## 1.13 Выражения и их вычисление

Разбор выражений производится слева направо с использованием обратной польской записи. Для изменения приоритета операций используются круглые скобки в выражении не может быть больше одного знака сравнения.

## 1.14 Конструкции языка

Конструкции языка TOV-2022 приведены в таблице 1.5.

Таблица 1.5 – конструкции языка TOV-2022

|  |  |
| --- | --- |
| Конструкции | Представление в языке |
| Главная функция | main |
| Пользовательская функция | <тип данных> function <идентификатор>([ (<тип данных><идентификатор>)\*]){…  return[<идентификатор>|<литерал>];} |
| Подключаемая библиотечная функция | add <тип данных> function <идентификатор> (>([ (<тип данных><идентификатор>)\*]); |
| Объявление переменной | var <тип данных> <идентификатор>; |

Пример подключения библиотечной функции stringlength:

add string function stringlength(string str);

Пример функции main:

main

{

var string a;

a = ‘qwerty’;

print a + ‘zxc’;

}

## 1.15 Область видимости идентификаторов

Каждая функция имеет свою область видимости, в которой располагаются её переменные и параметры. Переменные (но не литералы) из разных областей видимости не могут использоваться вместе. В глобальной области видимости могут находиться только функции. Литералы, как и параметры, принадлежат к областям видимости тех функций, в которых они были впервые объявлены, однако поиск литерала в таблице идентификаторов не учитывает область видимости, поэтому дублирования констант не происходит.

## 1.16 Семантические проверки

Список семантических проверок приведён в таблице 1.6

Таблица 1.6 – семантические проверки языка TOV-2022

|  |  |
| --- | --- |
| 1 | Наличие точки входа (main) |
| 2 | Единственность точки входа |
| 3 | Переобъявление идентификаторов |
| 4 | Правильность типов в выражениях (запрет на вычитание, умножение и деление строк и символов) |
| 5 | Правильность передаваемых в функцию параметров: количество, типы |

## 1.17 Распределение оперативной памяти на этапе выполнения

Все переменные размещаются в куче.

## 1.18 Стандартная библиотека и её состав

Необходимые функции стандартной библиотеки, недоступные пользователю, автоматически подключаются при компиляции. Доступные пользователю функции представлены в таблице 1.7.

Таблица 1.7 – доступные пользователю функции стандартной библиотеки

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификатор | Тип | Параметры | Описание |
| userstringcompare | number | string, string | Возвращает -1, если первая строка меньше второй, 1, если вторая больше первой, и 0, если строки равны |
| tonumber | number | string | Возвращает -1, если в строке присутствуют символы, не являющиеся цифрами, в противном случае возвращает число, составленное из исходной строки |
| indexof | number | string, string | Возвращает индекс начала подстроки в строке. В случае отсутствия подстроки возвращает -1 |

## 1.19 Ввод и вывод данных

Вывод данных предусмотрен с использованием ключевого слова print, осуществляющего вывод строки в диалоговое окно, ввод не предусмотрен.

## 1.20 Точка входа

Единственной и обязательной точкой входа является ключевое слово main.

## 1.21 Препроцессор

Использование препроцессора не предусмотрено.

## 1.22 Соглашения о вызовах

В качестве соглашения о вызове используется стандартное соглашение stdcall. Параметры передаются через стек в обратном порядке.

## 1.23 Объектный код

Трансляция производится в код на языке Ассемблера.

## 1.24 Классификация сообщений транслятора

Коды сообщений об ошибках приведены в таблице 1.8.

Таблица 1.8 – коды сообщений об ошибках

|  |  |
| --- | --- |
| Интервал | Описание |
| 2-50 | Лексические ошибки |
| 51-99 | Синтаксические ошибки |

Продолжение таблицы 1.8

|  |  |
| --- | --- |
| Интервал | Описание |
| 100-149 | Недопустимый символ в исходном коде |
| 150-199 | Семантические ошибки |
| 0 | Недопустимый код ошибки |
| 1 | Системный сбой |

## 1.25 Контрольный пример

Контрольный пример представлен в приложении А

**Глава 2. Структура транслятора**

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

Транслятор языка программирования TOV-2022 состоит из следующих частей:

Лексический анализатор – компонент транслятора, реализующий первую фазу работы компилятора. Он выделяет в входной последовательности символов языка простейшие конструкции языка (лексические единицы, или токены). На данном этапе распознаётся правильность составления и частично расположения лексем и идентификаторов.

Синтаксический анализатор – компонент транслятора, выполняющий синтаксический анализ. Он предназначен для распознавания синтаксических конструкций и формирования промежуточного кода. Проверяется правильность инструкций исходного кода. На вход принимает таблицу лексем.

Семантический анализатор – компонент транслятора, предназначенный для распознавания синтаксических конструкций и формирования промежуточного кода. На вход принимает таблицу лексем и таблицу идентификаторов.

Генератор кода – компонент транслятора, выполняющий генерацию ассемблерного кода на основе полученных на предыдущих этапах данных трансляции. На вход принимает таблицу лексем и таблицу идентификаторов, на выход подаёт файл с ассемблерным кодом.

**2.2 Перечень входных параметров транслятора**

На вход транслятора подаются параметры, представленные в таблице 2.1.

Таблица 2.1 – перечень входных параметров транслятора

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание параметра | Значение по умолчанию |
| -in:<имя in-файла> | Файл с исходным кодом на языке TOV-2022 | Отсутствует, в случае отсутствия параметра генерируется ошибка |
| -log:<имя log-файла> | Файл, в который будет записываться протокол работы транслятора | <имя in-файла>.log |

**2.3 Протоколы, формируемые транслятором**

В ходе работы транслятора формируются таблицы идентификаторов и лексем и дерево разбора, формирующие протокол работы транслятора, который сохраняется в файле .log.

# **Глава 3. Разработка лексического анализатора**

**3.1 Структура лексического анализатора**

Лексический анализатор – компонент транслятора, реализующий первую фазу работы компилятора. Он выделяет в входной последовательности символов языка простейшие конструкции языка (лексические единицы, или токены). На данном этапе распознаётся правильность составления и частично расположения лексем и идентификаторов. На вход принимает исходный код на языке TOV-2022. На выход подаёт таблицу идентификаторов и таблицу лексем.

Структура лексического анализатора представлена на рисунке 3.1.

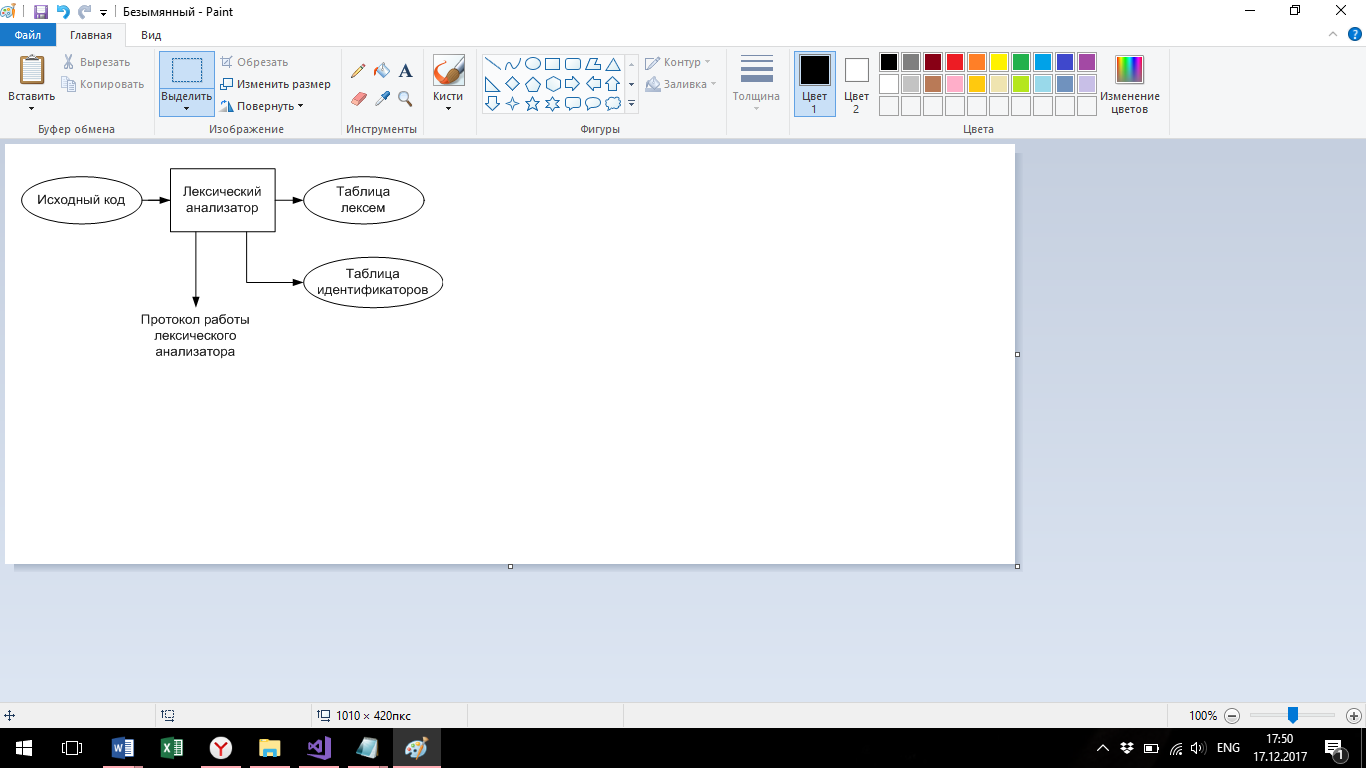


Рисунок 3.1 Структура лексического анализатора

**3.2 Контроль входных символов**

Все символы в кодировке Windows-1251 разбиваются на категории, представленные в таблице 3.1.

Таблица 3.1 – соответствие символов и их значений в таблице

|  |  |
| --- | --- |
| Значение в таблице входных символов | Символы |
| Разрешенный | T |
| Запрещенный | F |
| Игнорируемый | I |
| Сепаратор | S |

**3.3 Удаление избыточных символов**

Ненужные для трансляции символы удаляются. Так, символы табуляции и перевода строки игнорируются полностью, а пробелы учитываются только в том случае, если они являются частью строкового литерала.

**3.4 Перечень ключевых слов**

Соответствие ключевых слов, сепараторов, символов операций с лексемами приведено в таблице 3.2.

Таблица 3.2 – соответствие ключевых слов и сепараторов с лексемами

|  |  |  |  |
| --- | --- | --- | --- |
| Конструкция | | Лексема | Примечание |
| number  string | | t | Названия типов данных языка. |
| <Идентификатор> | | i | Длина идентификатора – 9 символов. |
| <Литерал> | | l | Литерал любого доступного типа. |
| func | | f | Объявление функции. |
| return | | r | Выход из функции. |
| main | | m | Главная функция. |
| var | | n | Объявление переменной. |
| print | | p | Вывод данных. |
| add | | a | Подключение библиотечной функции |
| if | | c | Начало условной инструкции |
| else | | e | Начало инструкции «иначе» |
| ; | | ; | Разделение инструкций. |
| , | | , | Разделение параметров функций. |
| { | | { | Начало блока/тела функции. |
| } | | } | Закрытие блока/тела функции. |
| ( | | ( | Передача параметров в функцию, приоритет операций. |
| ) | | ) | Закрытие блока для передачи параметров, приоритет операций. |
| = | | w | Знак присваивания. |
| + – \* / | v | | Знаки арифметических операций |
| < > <= >= == != | u | | Знаки операций сравнения |

Реализация графов переходов находится в приложении Б.

Пример реализованного конечного автомата ключевого слова main языка TOV-2022:

'm', FSTN::FSTsmall

{

5,

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

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

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

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

FSTN::NODE()

}

**3.5 Основные структуры данных**

Основные структуры данных приведены в приложении В.

Таблица лексем содержит исходный код, каждая лексема которого преобразована в структуру, хранящую информацию о себе. Назначение полей элементов таблицы лексем представлено в таблице 3.3.

Таблица 3.3 – назначение полей элементов таблицы лексем

|  |  |
| --- | --- |
| Поле | Назначение |
| lexema | Содержит символьное обозначение лексемы, отражающее её роль в исходном коде |
| sn | Содержит номер строки, на которой находится лексема. Используется для отображения ошибок |
| idxTi | Содержит индекс элемента в таблице идентификаторов, на который ссылается лексема. Если лексема ни на что не ссылается, содержит -1 |
| priority | Содержит значение приоритета, используемого знаками операций при построении обратной польской записи. Также используется, чтобы хранить количество параметров, передаваемых в функцию, если лексемой является оператор вызова функции в обратной польской записи |
| params | Содержит вектор перечислений IDDATATYPE, служит для проверки количества параметров и соответствия из типов при вызове функций |
| signtype | Содержит перечисление SIGNTYPE, используется при проверке допустимости данной операции для данных операндов |
| datatype | Содержит перечисление IDDATATYPE, используется для проверки правильности возвращаемого из функции типа |

Таблица идентификаторов содержит записи о каждом уникальном идентификаторе: функциях, переменных и литералах. Назначение полей элементов таблицы идентификаторов представлено в таблице 3.4.

Таблица 3.4 – назначение полей элементов таблицы лексем

|  |  |
| --- | --- |
| Поле | Назначение |
| idxfirstLE | Содержит индекс элемента в таблице лексем, на который ссылается данный идентификатор |
| id | Содержит уникальный идентификатор |
| iddatatype | Содержит перечисление IDDATATYPE, используется при проверке типов |
| idtype | Содержит перечисление IDTYPE, используется при проверке правильности использования идентификатора |
| scope | Содержит область видимости идентификатора |
| value | Содержит объединение, содержащее значение идентификатора |

**3.6 Структура и перечень сообщений лексического анализатора**

Структура ошибки содержит её номер, номер строки и номер символа, где было вызвано сообщение в исходном коде, а также текстовое пояснение. Перечень сообщений представлены в таблице 3.5.

Таблица 3.5 – перечень ошибок лексического анализатора

|  |  |
| --- | --- |
| Код сообщения | Содержание сообщения |
| 3 | Размер таблицы идентификаторов превышает макисмальный |
| 4 | Размер таблицы лексем превышает максимальный |
| 5 | Идентификатор совпадает с ключевым словом |
| 6 | Необходимо указать тип переменной при первом объявлении |
| 7 | Неизвестный тип идентификатора |
| 8 | Необходимо указать тип параметра при использовании нового идентификатора |
| 9 | Индекс n в функции GetEntry (idtable) вне допустимых пределов |
| 10 | Индекс n в функции GetEntry (lextable) вне допустимых пределов |
| 11 | Неизвестная лексема |
| 12 | Превышен лимит литералов: 999 |

Продолжение таблицы 3.5

|  |  |
| --- | --- |
| 13 | Знак '!' может использваться только как часть оператора '!=' |
| 14 | Идентификатор функции не может быть короче 3 символов |
| 15 | Попытка объявить библиотечную функцию в качестве переменной или параметра |
| 16 | Размер строки превышает максимальный |

**3.7 Принцип обработки ошибок**

В случае возникновения ошибок происходит их протоколирование с номером ошибки и сообщением. Лексический анализатор прекращает работу с исходным кодом и выводит оставшиеся сообщения об ошибках в файл протокола, если они существуют.

**3.8 Параметры лексического анализатора**

Параметром лексического анализатора является массив символов исходного кода, проверенного на допустимость символов.

**3.9 Алгоритм лексического анализа**

Алгоритм работы лексического анализа заключается в распознавании и разборе цепочек исходного кода на основе конечных автоматов, а также заполнение таблиц идентификаторов и лексем. Работу конечного автомата можно показать с помощью графа переходов. Пример графа для цепочки «main» приведен на рисунке 3.3.



Рисунок 3.3 Пример графа для цепочки main

В зависимости от того, каким автоматом была разобрана лексема, лексический анализатор выполняет различные действия.

Если разобран идентификатор, анализатор ищет в таблице идентификаторов элемент с таким же идентификатором и областью видимости, проверяет на совпадение с ключевыми словами и недоступными пользователю библиотечными функциями. Он вычисляет тип идентификатора и проверяет идентификатор на допустимость использования в текущем контексте (функция не может передаваться в качестве параметра, идентификатор, зарезервированный для функции, нельзя объявить в качестве переменной).

Если разобран литерал, анализатор ищет литерал с таким же значением в таблице идентификаторов, и, если его не существует, добавляет текущий литерал в таблицу идентификаторов.

**3.10 Контрольный пример**

Контрольный пример в виде таблицы лексем и таблицы идентификаторов представлен в приложении Г.

**Глава 4. Разработка синтаксического анализатора**

**4.1 Структура синтаксического анализатора**

Синтаксический анализатор – второй компонент транслятора, вызываемый после завершения работы лексического анализатора. Он принимает на вход результат работы лексического анализатора: таблицу лексем и таблицу идентификаторов – и осуществляет проверку преобразованного кода на соответствие правилам синтаксиса языка. На выход синтаксический анализатор подаёт дерево разбора.

* 1. **Контекстно-свободная грамматика, описывающая синтаксис языка**

Синтаксис языка TOV-2022 описывается грамматикой типа 2 по иерархии Хомского:

G = <T, N, P, S>

T – множество терминальных символов (алфавит языка TOV-2022),

N – множество нетерминальных символов,

P – множество правил языка,

S – начальный символ грамматики, представленный нетерминальным символом «S».

Множество терминальных символов приведено в таблице 3.2. Множество нетерминальных символов и соответствующие им правила приведены в таблице 4.1.

Таблица 4.1 – нетерминалы и соответствующие им правила

|  |  |  |
| --- | --- | --- |
| Нетерминал | Назначение | Правила |
| S | Общий вид программы | m{N}Z |
| m{N} |
| tfi(P)BZ |
| tfi(P)B |
| atfi(P);Z |
| atfi(P); |
| tfi()BZ |
| tfi()B |
| atfi();Z |
| atfi(); |
| Z | Общий вид программы | m{N}Z |
| m{N} |
| tfi(P)BZ |
| tfi(P)B |
| atfi(P);Z |
| atfi(P); |
| tfi()BZ |
| tfi()B |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Нетерминал | Назначение | Правила |
|  |  | atfi();Z |
|  |  | atfi(); |
| B | Тело функции | NrE; |
| rE; |
| N | Возможные инструкции | nti; |
| cCA |
| i = E; |
| ntfi(F); |
| nti;N |
| cCAN |
| i = E;N |
| ntfi(F);N |
| pE; |
| pE;N |
| E | Правила выражений | i |
| i() |
| i()M |
| i(W)M |
| (E)M |
| lM |
| iM |
| i(W) |
| (E) |
| l |
| M | Правила выражений | vE |
| vEM |
| P | Правила для параметров вызываемых  функций | ti |
| ti,P |
| W | Правила для аргументов | i |
| l,W |
| i,W |
| l |
| A | Условная конструкция | {D}Ge{D} |
| {D}G |
| {D}e{D} |
| {D} |
| D | Условный блок | iwE;D |
| iwE; |
| pE; |
| pE;D |
| G | Блоки else и else if | ecC{D} |
| ecC{D}G |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Нетерминал | Назначение | Правила |
| C | Условие | (EuE) |

**4.3 Построение конечного магазинного автомата**

Распознавателем грамматики является конечный автомат с магазинной памятью, который представляет собой семерку. Подробное описание компонентов магазинного автомата представлено в таблице 4.2.

Таблица 4.2 – описание составных частей магазинного автомата

|  |  |
| --- | --- |
| Компонента | Определение |
|  | Множество состояний автомата |
|  | Алфавит входных символов |
|  | Алфавит специальных магазинных символов |
|  | Функция переходов автомата |
|  | Начальное состояние автомата |
|  | Начальное состояние магазина автомата |
|  | Множество конечных состояний |

Схема МП-автомата представлена на рисунке 4.1

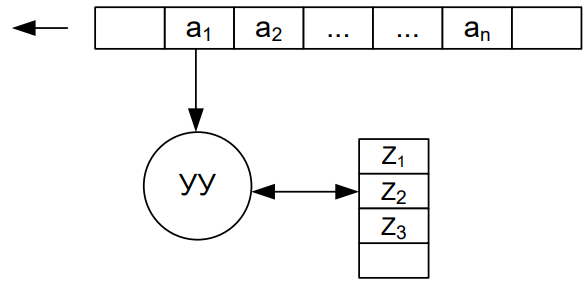


Рисунок 4.1 Схема МП-автомата

Пример последовательности мгновенных состояний МП-автомата, демонстрирующую успешный разбор для цепочки языка m{nti;iwlv(lvl);pi;}: S -> m{N} -> m{nti;N} -> m{nti;iwE;N} -> m{nti;iwlM;N} -> m{nti;iwl(E);N} -> m{nti;iwl(lM);N} -> m{nti;iwl(lM);N} -> m{nti;iwl(lvl);N} -> m{nti;iwl(lvl);pi}.

**4.4 Основные структуры данных**

Контекстно-свободная грамматика описывается структурой Greibach, представленной в приложении Д.

**4.5 Описание алгоритма синтаксического разбора**

Алгоритм синтаксического разбора можно описать следующим образом:

* В магазин добавляются конечный и начальный символы.
* Таблица лексем образует входную ленту.
* Выбирается первая цепочка из терминальных и нетерминальных символов, соответствующая данному нетерминальному символу.
* Если на вершине магазина и ленты лежат одинаковые терминальные символы, они удаляются из магазина и ленты. Иначе происходит откат до предыдущего сохраненного состояния и выбор другой цепочки нетерминального символа.
* Если в магазине встретился нетерминал, происходит переход к пункту 3.
* Если на вершине магазина и ленты лежит конечный символ, синтаксический разбор выполнен успешно, в противном случае генерируется ошибка.

**4.6 Структура и перечень сообщений синтаксического анализатора**

Перечень сообщений синтаксического анализатора представлен в таблице 4.3.

Таблица 4.3 – перечень ошибок синтаксического анализатора

|  |  |
| --- | --- |
| Код сообщения | Содержание сообщения |
| 50 | На верхнем уровне должны находиться только функции |
| 52 | Функция обязана возвращать значение |
| 53 | Недопустимая инструкция в теле функции |
| 54 | Неверно составленное выражение |
| 55 | Неверное арифметическое выражение |
| 56 | Неверный список параметров |
| 57 | Неверный список аргументов |
| 58 | Неверно составленное условное выражение |
| 59 | В условном блке могут содержаться только операции присваивания и вывода |
| 60 | Неверно составленные блоки else if |
| 61 | Условием может являться только сравнение двух выражений |
| 63 | Неизвестная синтаксическая ошибка |
| 64 | Отсутствует подходящее правило |
| 65 | Отсутствует подходящая цепочка правил |

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

Входными параметрами для синтаксического анализатора в языке программирования TOV-2022 являются таблица лексем и таблица идентификаторов.

**4.8 Принцип обработки ошибок**

Синтаксический анализатор перебирает все возможные правила и цепочки правила грамматики, начинающиеся на верхний символ ленты, в целях поиска подходящего соответствия. Если ни одна из цепочек правила не подошла для рассматриваемой конструкции, то генерируется ошибка в соответствии с таблицей 4.3. Ошибка заносится в протокол.

**4.9 Контрольный пример**

Пример разбора исходного кода на языке программирования TOV-2022 синтаксическим анализатором представлен в приложении Е.

**Глава 5. Разработка семантического анализатора**

**5.1 Структура семантического анализатора**

Семантический анализатор принимает на вход таблицу лексем и таблицу идентификаторов и ищет в них ошибки. Также некоторые проверки, например, на правильность объявления переменных и их повторное объявление, производятся на стадии лексического анализа

**5.2 Функции семантического анализатора**

Семантический анализатор выполняет проверку на основе правил языка, представленных в таблице 1.6.

**5.3 Структура и перечень сообщений семантического анализатора**

Список сообщений об ошибках, генерируемых семантическим анализатором, представлен в таблице 5.1.

Таблица 5.1 – перечень ошибок семантического анализатора

|  |  |
| --- | --- |
| Код сообщения | Содержание сообщения |
| 150 | Отсутствует точка входа |
| 151 | Точек входа больше одной |
| 152 | Несоответствие типа при присваивании |
| 153 | Несоответствие типа параметра |
| 154 | Превышено число параметров |
| 155 | Недостаточно параметров |
| 156 | Не найдено функции с данным идентификатором |
| 157 | Функция должна вызываться с параметрами |
| 158 | В операции сравнения не может быть больше одного оператора сравнения |
| 159 | В операции сравнения обязан присутствовать оператор сравнения |
| 160 | Несоответствие типа в операции сравнения |
| 161 | Из арифметических операций к строкам и символам можно применять только сложение |
| 162 | Присваивать можно только идентификаторам |
| 163 | В инструкции до знака присваивания может стоять только идентификатор |
| 164 | В операциях сравнения могут участвовать только числа |
| 165 | Несоответствие типа параметра библиотечной функции |
| 166 | Несоответствие возвращаемого типа библиотечной функции |

**5.4 Принцип обработки ошибок**

Семантический анализатор просматривает таблицу лексем, в случае обнаружения ключевых слов или знаков проверяет правильность инструкций, при обнаружении ошибки выполнение трансляции прекращается, ошибка заносится в протокол.

**5.5 Контрольный пример**

Обработка ошибок семантического анализатора представлена в таблице 8.4.

**Глава 6. Преобразование выражений**

**6.1 Выражения, допускаемые языком**

В языке программирования TOV-2022 выражения могут содержать вычисления целочисленных типов данных, а также допускаются вызов функций (возвращающих тип) внутри выражений. Приоритет операций представлен в таблице 1.4

**6.2 Польская запись и принцип ее построения**

В языке программирования TOV-2022 представлена обратная польская запись. В процессе преобразования операнды и операторы меняют свой порядок. Результатом вычисления становится результат последней вычисленной операции.

Принцип построения обратной польской записи:

* исходная строка: выражение;
* результирующая строка: польская запись;
* стек: пустой;
* 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;
* отрывающая скобка помещается в стек;
* закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются;
* по концу разбора исходной строки все операции, оставшиеся в стеке, выталкиваются в результирующую строку.
* если в исходном выражении присутствуют скобки, результирующая строка заполняется лексемами-заглушками, позволяющими не сдвигать влево все элементы таблицы лексем, находящиеся после выражения

Примеры преобразования в обратную польскую нотацию приведены в таблице 6.1.

Таблица 6.1 – Примеры преобразования выражений из исходного кода в обратную польскую нотацию

|  |  |
| --- | --- |
| Фрагмент таблицы лексем до преобразования | Фрагмент таблицы лексем после преобразования |
| iv(ivi) | iiivv |
| iui | iiu |
| ivliv | ilviv |

**6.3 Программная реализация обработки выражений**

Программная реализация обработки выражений представлена в приложении Ж.

**6.4 Контрольный пример**

Пример преобразования выражения к обратной польской записи представлен на рисунке 6.1.

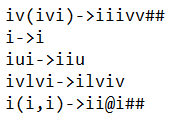


Рисунок 6.1 Пример преобразования выражения к обратной польской записи

**Глава 7. Генерация кода**

## 7.1 Структура генератора кода

Генератор кода принимает на вход таблицу лексем и таблицу идентификаторов. В соответствии с таблицей лексем строится выходной файл на языке ассемблера, который является результатом работы генератора. Структура генератора кода TOV-2022 представлена на рисунке 7.1.

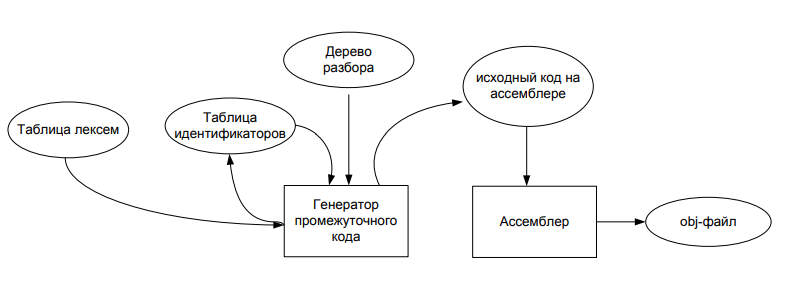


Рисунок 7.1 Структура генератора кода

## 7.2 Представление типов данных в оперативной памяти

Элементы таблицы идентификаторов расположены в сегментах .DATA и .CONST языка ассемблера. Соответствия между типами данных идентификаторов на языке TOV-2022 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 Соответствия типов идентификаторов

|  |  |  |
| --- | --- | --- |
| Тип идентификатора | Тип идентификатора на языке ассемблера | Пояснение |
| number | sdword | Хранит целочисленный тип данных |
| string | dword | Хранит указатель на начало строки |
| symbol | dword | Хранит указатель на начало строки |
| L<тип><порядковый номер литерала> | sdword, dword | Целочисленные, символьные, булевы и строковые литералы |

## 7.3 Статическая библиотека

В языке TOV-2022 предусмотрена статическая библиотека, которая содержит функции лексикографического сравнения строк, преобразования строки в число, поиска подстроки в строке. Чтобы пользоваться данными функциями, пользователь должен их подключить.

## 7.4 Особенности алгоритма генерации кода

Алгоритм генерации объектного кода выглядит следующим образом:

Заполняется необходимая информация: модель процессора, модель памяти соглашение о вызовах, подключаются необходимые библиотеки, указывается размер стека и добавляются прототипы служебных функций.

Затем добавляются прототипы функций, которыми желает пользоваться пользователь. Генератор проходит по таблице лексем и, встречая лексему, отвечающую за ключевое слово function, вызывает функцию, добавляющую соответствующий прототип.

Далее генератор дважды проходит по таблице идентификаторов и заполняет поля переменных и констант идентификаторами, типами и значениями. Пример заполнения полей констант и переменных:

.CONST

Li000 SDWORD 0

Li001 SDWORD 8

Ly002 BYTE 1, "q"

Ly003 BYTE 1, "x"

Ls004 BYTE 0

Ls005 BYTE 14, " is less than "

Ls006 BYTE 8, " equals "

Ls007 BYTE 16, " is bigger than "

Ls008 BYTE 7, "1234432"

.DATA

fii@x SDWORD 0

fss@a BYTE 256 DUP(0)

fss@s BYTE 0

fss@z BYTE 0

main@x SDWORD 0

main@z SDWORD 0

main@sa BYTE 256 DUP(0)

Затем создаётся процедура main, по таблице лексем определяются её границы, процедура наполняется инструкциями и завершается вызовом функции завершения процесса.

Далее происходит преобразование пользовательских функций в язык Ассемблера.

## 7.5 Контрольный пример

Контрольный пример ассемблерного кода приведен в приложении З.

Результат генерации ассемблерного кода приведен на рисунке 7.2.

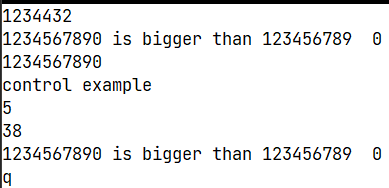


Рисунок 7.1 Структура генератора кода

**Глава 8. Тестирование транслятора**

## 8.1 Тестирование проверки на допустимость символов

В языке TOV-2022 не разрешается использовать запрещённые входным алфавитом символы. Результат обнаружения запрещённого символа представлен в таблице 8.1.

Таблица 8.1 Тестирование проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| x = ‘абвгдеёж’; | Запрещённый символ в строке 65 в столбце 7 код символа: 224  Error #111 Недопустимый символ в исходном файле (-in) at 8 column 66 line |

## 8.2 Тестирование лексического анализатора

Результаты тестирования лексического анализатора показаны в таблице 8.2.

Таблица 8.2 Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| x = tonumber('1234432'); | Error #7 Неизвестный тип идентификатора at 3 column 58 line |
| var string tonumber; | Error #15 Попытка объявить библиотечную функцию в качестве переменной или параметра at 21 column 3 line |
| var x; | Error #6 Необходимо указать тип переменной при первом объявлении at 7 column 3 line |

## 8.3 Тестирование синтаксического анализатора

Результаты тестирования синтаксического анализатора представлены в таблице 8.3.

Таблица 8.3 Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  {  var number x;  return x;  } | Error #64 Отсутствует подходящее правило, подробности в log-файле  Log-файл: 53: line 4, Недопустимая инструкция в теле функции, r  53: line 4, Недопустимая инструкция в теле функции, r  53: line 3, Недопустимая инструкция в теле функции, n |

Продолжение таблицы 8.3

|  |  |
| --- | --- |
| if (1){print 1;}  else{print 2;} | Error #64 Отсутствует подходящее правило, подробности в log-файле  Log-файл: 55: line 4, Неверное арифметическое выражение, ) |

## 8.4 Тестирование семантического анализатора

Итоги тестирования семантического анализатора приведены в таблице 8.4.

Таблица 8.4 Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| string function fff()return 'qwerty';} | Error #150 Отсутствует точка входа |
| main{print 1;}  main{print 2;} | Error #151 Точек входа больше одной |
| main{print 'qwerty' - 'qwe';} | Error #161 Из арифметических операций к строкам и символам можно применять только сложение at 0 column 1 line |

**Заключение**

В ходе выполнения курсового проектирования был разработан компилятор языка программирования TOV-2022 в язык Ассемблера. Таким образом, были выполнены основные задачи данной курсовой работы:

* создана спецификация языка TOV-2022;
* разработана структура транслятора;
* разработан лексический анализатор;
* разработан синтаксический анализатор;
* разработан семантический анализатор;
* разработаны функции преобразования выражений;
* разработан генератор кода на языке Ассемблера;
* проведено тестирование транслятора на контрольных примерах.

Язык TOV-2022 имеет 3 типа данных, два из которых, однако, практически одинаковы, 5 инструкций: объявление переменной, присваивание, условная инструкция, вывод данных, возврат данных – 22 лексемы, 12 правил грамматики, стандартную библиотеку с четырьмя функциями для пользователя и семью служебными функциями. Было написано приблизительно 3616 строк кода на С++, трансляция контрольного примера заняла приблизительно 513 миллисекунд.

**Список использованных источников**

1 Ирвин, К. Р. Язык ассемблера для процессоров Intel, 4-е издание. Пер. с англ. – М.: Издательский дом «Вильямс», 2005. – 912 с.

2 Документация по языку C++ <https://learn.microsoft.com/ru-ru/cpp/cpp/?view=msvc-160>

3 Руководство по языку программирования C++ <https://metanit.com/cpp/tutorial/>

4 C++ Language <https://cplusplus.com>

5 Stack Overflow https://stackoverflow.com

**Приложение А**

Контрольный пример исходного кода на языке TOV-2022:

add number function tonumber(string a);

add number function stringlength(string a);

add number function indexof(string a, string b);

add number function userstringcompare(string a, string b);

number function fii(number x, number y)

{

var number z;

z = x \* (x + y);

return z;

}

string function fss (string a, string b)

{

var string c;

if (a < b)

{

c = a + ' is less than ' + b;

print indexof(a, b);

}

else if (a == b)

{

c = a + ' equals ' + b;

print stringlength(a);

}

else

{

c = a + ' is bigger than ' + b;

print tonumber(a);

}

return c;

}

main

{

var number x;

var number y;

var string sa;

var string sb;

var string sc;

x = 1;

y = 5;

sa = '1234567890';

sb = '123456789 0';

print fii(x, y);

sc = fss(sa, sb);

print sc;

print stringlength(sc);

print userstringcompare(sa, sb);

}

**Приложение Б**

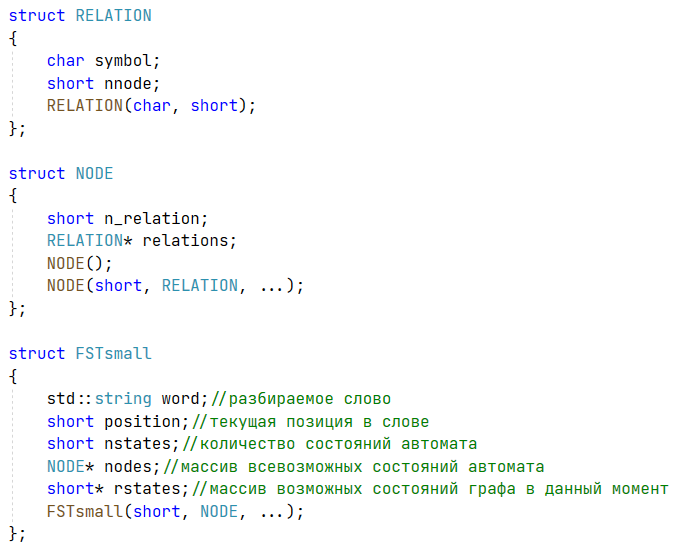


Рисунок Б.1 Реализация графов переходов

**Приложение В**

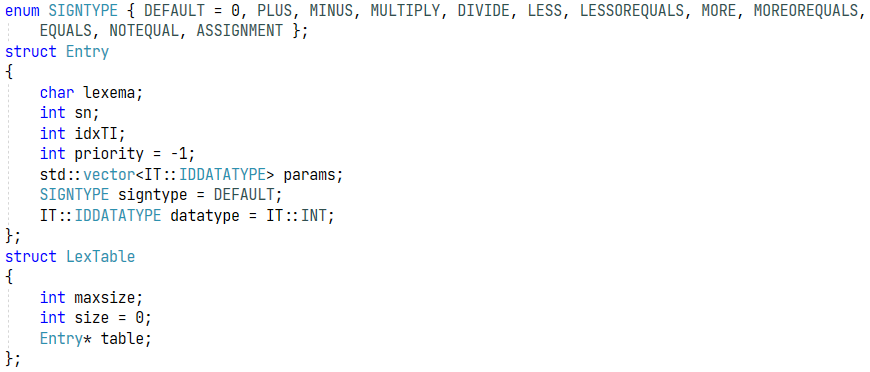


Рисунок В.2 – реализация таблицы лексем

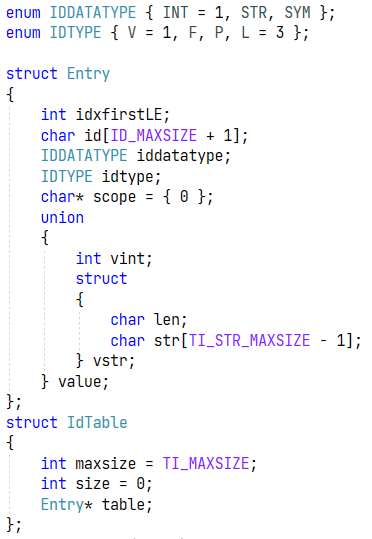


Рисунок В.3 Реализация таблицы идентификаторов

**Приложение Г**

Таблица идентификаторов:

0) tonumber | scope: | firstLE: 3 | INT | F | 0

1) stringlength | scope: | firstLE: 12 | INT | F | 0

2) indexof | scope: | firstLE: 21 | INT | F | 0

3) userstringcompare | scope: | firstLE: 33 | INT | F | 0

4) fii | scope: | firstLE: 44 | INT | F | 0

5) x | scope: fii | firstLE: 47 | INT | L/P | 0

6) y | scope: fii | firstLE: 50 | INT | L/P | 0

7) z | scope: fii | firstLE: 55 | INT | V | 0

8) fss | scope: | firstLE: 73 | STR | F | vstr.len: 0 | vstr.str:

9) a | scope: fss | firstLE: 76 | STR | L/P | vstr.len: 0 | vstr.str:

10) b | scope: fss | firstLE: 79 | STR | L/P | vstr.len: 0 | vstr.str:

11) c | scope: fss | firstLE: 84 | STR | V | vstr.len: 0 | vstr.str:

12) Ls000 | scope: fss | firstLE: 97 | STR | L/P | vstr.len: 14 | vstr.str: is less than

13) Ls001 | scope: fss | firstLE: 122 | STR | L/P | vstr.len: 8 | vstr.str: equals

14) Ls002 | scope: fss | firstLE: 139 | STR | L/P | vstr.len: 16 | vstr.str: is bigger than

15) x | scope: main | firstLE: 158 | INT | V | 0

16) y | scope: main | firstLE: 162 | INT | V | 0

17) sa | scope: main | firstLE: 166 | STR | V | vstr.len: 0 | vstr.str:

18) sb | scope: main | firstLE: 170 | STR | V | vstr.len: 0 | vstr.str:

19) sc | scope: main | firstLE: 174 | STR | V | vstr.len: 0 | vstr.str:

20) Li003 | scope: main | firstLE: 178 | INT | L/P | 1

21) Li004 | scope: main | firstLE: 182 | INT | L/P | 5

22) Ls005 | scope: main | firstLE: 186 | STR | L/P | vstr.len: 10 | vstr.str: 1234567890

23) Ls006 | scope: main | firstLE: 190 | STR | L/P | vstr.len: 12 | vstr.str: 123456789 0

Таблица лексем:

0) a | idxTI: -1 | string number: 1 | params:

1) t | idxTI: -1 | string number: 1 | params:

2) f | idxTI: -1 | string number: 1 | params:

3) i | idxTI: 0 | string number: 1 | params: str

4) ( | idxTI: -1 | string number: 1 | params:

5) t | idxTI: -1 | string number: 1 | params:

6) i | idxTI: -1 | string number: 1 | params:

7) ) | idxTI: -1 | string number: 1 | params:

8) ; | idxTI: -1 | string number: 1 | params:

9) a | idxTI: -1 | string number: 2 | params:

10) t | idxTI: -1 | string number: 2 | params:

11) f | idxTI: -1 | string number: 2 | params:

12) i | idxTI: 1 | string number: 2 | params: str

13) ( | idxTI: -1 | string number: 2 | params:

14) t | idxTI: -1 | string number: 2 | params:

15) i | idxTI: -1 | string number: 2 | params:

16) ) | idxTI: -1 | string number: 2 | params:

17) ; | idxTI: -1 | string number: 2 | params:

18) a | idxTI: -1 | string number: 3 | params:

19) t | idxTI: -1 | string number: 3 | params:

20) f | idxTI: -1 | string number: 3 | params:

21) i | idxTI: 2 | string number: 3 | params: str str

22) ( | idxTI: -1 | string number: 3 | params:

23) t | idxTI: -1 | string number: 3 | params:

24) i | idxTI: -1 | string number: 3 | params:

25) , | idxTI: -1 | string number: 3 | params:

26) t | idxTI: -1 | string number: 3 | params:

27) i | idxTI: -1 | string number: 3 | params:

28) ) | idxTI: -1 | string number: 3 | params:

29) ; | idxTI: -1 | string number: 3 | params:

30) a | idxTI: -1 | string number: 4 | params:

31) t | idxTI: -1 | string number: 4 | params:

32) f | idxTI: -1 | string number: 4 | params:

33) i | idxTI: 3 | string number: 4 | params: str str

34) ( | idxTI: -1 | string number: 4 | params:

35) t | idxTI: -1 | string number: 4 | params:

36) i | idxTI: -1 | string number: 4 | params:

37) , | idxTI: -1 | string number: 4 | params:

38) t | idxTI: -1 | string number: 4 | params:

39) i | idxTI: -1 | string number: 4 | params:

40) ) | idxTI: -1 | string number: 4 | params:

41) ; | idxTI: -1 | string number: 4 | params:

42) t | idxTI: -1 | string number: 6 | params:

43) f | idxTI: -1 | string number: 6 | params:

44) i | idxTI: 4 | string number: 6 | params: int int

45) ( | idxTI: -1 | string number: 6 | params:

46) t | idxTI: -1 | string number: 6 | params:

47) i | idxTI: 5 | string number: 6 | params:

48) , | idxTI: -1 | string number: 6 | params:

49) t | idxTI: -1 | string number: 6 | params:

50) i | idxTI: 6 | string number: 6 | params:

51) ) | idxTI: -1 | string number: 6 | params:

52) { | idxTI: -1 | string number: 7 | params:

53) n | idxTI: -1 | string number: 8 | params:

54) t | idxTI: -1 | string number: 8 | params:

55) i | idxTI: 7 | string number: 8 | params:

56) ; | idxTI: -1 | string number: 8 | params:

57) i | idxTI: 7 | string number: 9 | params:

58) w | idxTI: -1 | string number: 9 | params:

59) i | idxTI: 5 | string number: 9 | params:

60) v | idxTI: -1 | string number: 9 | params:

61) ( | idxTI: -1 | string number: 9 | params:

62) i | idxTI: 5 | string number: 9 | params:

63) v | idxTI: -1 | string number: 9 | params:

64) i | idxTI: 6 | string number: 9 | params:

65) ) | idxTI: -1 | string number: 9 | params:

66) ; | idxTI: -1 | string number: 9 | params:

67) r | idxTI: -1 | string number: 10 | params:

68) i | idxTI: 7 | string number: 10 | params:

69) ; | idxTI: -1 | string number: 10 | params:

70) } | idxTI: -1 | string number: 11 | params:

71) t | idxTI: -1 | string number: 12 | params:

72) f | idxTI: -1 | string number: 12 | params:

73) i | idxTI: 8 | string number: 12 | params: str str

74) ( | idxTI: -1 | string number: 12 | params:

75) t | idxTI: -1 | string number: 12 | params:

76) i | idxTI: 9 | string number: 12 | params:

77) , | idxTI: -1 | string number: 12 | params:

78) t | idxTI: -1 | string number: 12 | params:

79) i | idxTI: 10 | string number: 12 | params:

80) ) | idxTI: -1 | string number: 12 | params:

81) { | idxTI: -1 | string number: 13 | params:

82) n | idxTI: -1 | string number: 14 | params:

83) t | idxTI: -1 | string number: 14 | params:

84) i | idxTI: 11 | string number: 14 | params:

85) ; | idxTI: -1 | string number: 14 | params:

86) c | idxTI: -1 | string number: 15 | params:

87) ( | idxTI: -1 | string number: 15 | params:

88) i | idxTI: 9 | string number: 15 | params:

89) u | idxTI: -1 | string number: 15 | params:

90) i | idxTI: 10 | string number: 15 | params:

91) ) | idxTI: -1 | string number: 15 | params:

92) { | idxTI: -1 | string number: 16 | params:

93) i | idxTI: 11 | string number: 17 | params:

94) w | idxTI: -1 | string number: 17 | params:

95) i | idxTI: 9 | string number: 17 | params:

96) v | idxTI: -1 | string number: 17 | params:

97) l | idxTI: 12 | string number: 17 | params:

98) v | idxTI: -1 | string number: 17 | params:

99) i | idxTI: 10 | string number: 17 | params:

100) ; | idxTI: -1 | string number: 17 | params:

101) p | idxTI: -1 | string number: 18 | params:

102) i | idxTI: 2 | string number: 18 | params:

103) ( | idxTI: -1 | string number: 18 | params:

104) i | idxTI: 9 | string number: 18 | params:

105) , | idxTI: -1 | string number: 18 | params:

106) i | idxTI: 10 | string number: 18 | params:

107) ) | idxTI: -1 | string number: 18 | params:

108) ; | idxTI: -1 | string number: 18 | params:

109) } | idxTI: -1 | string number: 19 | params:

110) e | idxTI: -1 | string number: 20 | params:

111) c | idxTI: -1 | string number: 20 | params:

112) ( | idxTI: -1 | string number: 20 | params:

113) i | idxTI: 9 | string number: 20 | params:

114) u | idxTI: -1 | string number: 20 | params:

115) i | idxTI: 10 | string number: 20 | params:

116) ) | idxTI: -1 | string number: 20 | params:

117) { | idxTI: -1 | string number: 21 | params:

118) i | idxTI: 11 | string number: 22 | params:

119) w | idxTI: -1 | string number: 22 | params:

120) i | idxTI: 9 | string number: 22 | params:

121) v | idxTI: -1 | string number: 22 | params:

122) l | idxTI: 13 | string number: 22 | params:

123) v | idxTI: -1 | string number: 22 | params:

124) i | idxTI: 10 | string number: 22 | params:

125) ; | idxTI: -1 | string number: 22 | params:

126) p | idxTI: -1 | string number: 23 | params:

127) i | idxTI: 1 | string number: 23 | params:

128) ( | idxTI: -1 | string number: 23 | params:

129) i | idxTI: 9 | string number: 23 | params:

130) ) | idxTI: -1 | string number: 23 | params:

131) ; | idxTI: -1 | string number: 23 | params:

132) } | idxTI: -1 | string number: 24 | params:

133) e | idxTI: -1 | string number: 25 | params:

134) { | idxTI: -1 | string number: 26 | params:

135) i | idxTI: 11 | string number: 27 | params:

136) w | idxTI: -1 | string number: 27 | params:

137) i | idxTI: 9 | string number: 27 | params:

138) v | idxTI: -1 | string number: 27 | params:

139) l | idxTI: 14 | string number: 27 | params:

140) v | idxTI: -1 | string number: 27 | params:

141) i | idxTI: 10 | string number: 27 | params:

142) ; | idxTI: -1 | string number: 27 | params:

143) p | idxTI: -1 | string number: 28 | params:

144) i | idxTI: 0 | string number: 28 | params:

145) ( | idxTI: -1 | string number: 28 | params:

146) i | idxTI: 9 | string number: 28 | params:

147) ) | idxTI: -1 | string number: 28 | params:

148) ; | idxTI: -1 | string number: 28 | params:

149) } | idxTI: -1 | string number: 29 | params:

150) r | idxTI: -1 | string number: 30 | params:

151) i | idxTI: 11 | string number: 30 | params:

152) ; | idxTI: -1 | string number: 30 | params:

153) } | idxTI: -1 | string number: 31 | params:

154) m | idxTI: -1 | string number: 33 | params:

155) { | idxTI: -1 | string number: 34 | params:

156) n | idxTI: -1 | string number: 35 | params:

157) t | idxTI: -1 | string number: 35 | params:

158) i | idxTI: 15 | string number: 35 | params:

159) ; | idxTI: -1 | string number: 35 | params:

160) n | idxTI: -1 | string number: 36 | params:

161) t | idxTI: -1 | string number: 36 | params:

162) i | idxTI: 16 | string number: 36 | params:

163) ; | idxTI: -1 | string number: 36 | params:

164) n | idxTI: -1 | string number: 37 | params:

165) t | idxTI: -1 | string number: 37 | params:

166) i | idxTI: 17 | string number: 37 | params:

167) ; | idxTI: -1 | string number: 37 | params:

168) n | idxTI: -1 | string number: 38 | params:

169) t | idxTI: -1 | string number: 38 | params:

170) i | idxTI: 18 | string number: 38 | params:

171) ; | idxTI: -1 | string number: 38 | params:

172) n | idxTI: -1 | string number: 39 | params:

173) t | idxTI: -1 | string number: 39 | params:

174) i | idxTI: 19 | string number: 39 | params:

175) ; | idxTI: -1 | string number: 39 | params:

176) i | idxTI: 15 | string number: 40 | params:

177) w | idxTI: -1 | string number: 40 | params:

178) l | idxTI: 20 | string number: 40 | params:

179) ; | idxTI: -1 | string number: 40 | params:

180) i | idxTI: 16 | string number: 41 | params:

181) w | idxTI: -1 | string number: 41 | params:

182) l | idxTI: 21 | string number: 41 | params:

183) ; | idxTI: -1 | string number: 41 | params:

184) i | idxTI: 17 | string number: 42 | params:

185) w | idxTI: -1 | string number: 42 | params:

186) l | idxTI: 22 | string number: 42 | params:

187) ; | idxTI: -1 | string number: 42 | params:

188) i | idxTI: 18 | string number: 43 | params:

189) w | idxTI: -1 | string number: 43 | params:

190) l | idxTI: 23 | string number: 43 | params:

191) ; | idxTI: -1 | string number: 43 | params:

192) p | idxTI: -1 | string number: 44 | params:

193) i | idxTI: 4 | string number: 44 | params:

194) ( | idxTI: -1 | string number: 44 | params:

195) i | idxTI: 15 | string number: 44 | params:

196) , | idxTI: -1 | string number: 44 | params:

197) i | idxTI: 16 | string number: 44 | params:

198) ) | idxTI: -1 | string number: 44 | params:

199) ; | idxTI: -1 | string number: 44 | params:

200) i | idxTI: 19 | string number: 45 | params:

201) w | idxTI: -1 | string number: 45 | params:

202) i | idxTI: 8 | string number: 45 | params:

203) ( | idxTI: -1 | string number: 45 | params:

204) i | idxTI: 17 | string number: 45 | params:

205) , | idxTI: -1 | string number: 45 | params:

206) i | idxTI: 18 | string number: 45 | params:

207) ) | idxTI: -1 | string number: 45 | params:

208) ; | idxTI: -1 | string number: 45 | params:

209) p | idxTI: -1 | string number: 46 | params:

210) i | idxTI: 19 | string number: 46 | params:

211) ; | idxTI: -1 | string number: 46 | params:

212) p | idxTI: -1 | string number: 47 | params:

213) i | idxTI: 1 | string number: 47 | params:

214) ( | idxTI: -1 | string number: 47 | params:

215) i | idxTI: 19 | string number: 47 | params:

216) ) | idxTI: -1 | string number: 47 | params:

217) ; | idxTI: -1 | string number: 47 | params:

218) p | idxTI: -1 | string number: 48 | params:

219) i | idxTI: 3 | string number: 48 | params:

220) ( | idxTI: -1 | string number: 48 | params:

221) i | idxTI: 17 | string number: 48 | params:

222) , | idxTI: -1 | string number: 48 | params:

223) i | idxTI: 18 | string number: 48 | params:

224) ) | idxTI: -1 | string number: 48 | params:

225) ; | idxTI: -1 | string number: 48 | params:

226) } | idxTI: -1 | string number: 49 | params:

227) $ | idxTI: -1 | string number: -1 | params:

**Приложение Д**

Структура Geibach:

Greibach greibachOriginal(NS('S'), TS('$'), 12,

//S -> m{N}Z | m{N} | tfi(P)BZ | tfi(P)B | atfi(P);Z | atfi(P); | tfi()BZ | tfi()B | atfi();Z

// | atfi();

Rule(NS('S'), GRB\_ERRORS\_START, 10,// Неверная структура программы

Rule::Chain(5, TS('m'), TS('{'), NS('N'), TS('}'), NS('Z')),

Rule::Chain(4, TS('m'), TS('{'), NS('N'), TS('}')),

Rule::Chain(8, TS('t'), TS('f'), TS('i'), TS('('), NS('P'), TS(')'), NS('B'), NS('Z')),

Rule::Chain(7, TS('t'), TS('f'), TS('i'), TS('('), NS('P'), TS(')'), NS('B')),

Rule::Chain(7, TS('t'), TS('f'), TS('i'), TS('('), TS(')'), NS('B'), NS('Z')),

Rule::Chain(6, TS('t'), TS('f'), TS('i'), TS('('), TS(')'), NS('B')),

Rule::Chain(9, TS('a'), TS('t'), TS('f'), TS('i'), TS('('), NS('P'), TS(')'), TS(';'), NS('Z')),

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

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

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

),

//Z -> mBZ | mB | tfi(P)BZ | tfi(P)B | atfi(P);Z | atfi(P);

Rule(NS('Z'), GRB\_ERRORS\_START, 10,// Неверная структура программы

Rule::Chain(5, TS('m'), TS('{'), NS('N'), TS('}'), NS('Z')),

Rule::Chain(4, TS('m'), TS('{'), NS('N'), TS('}')),

Rule::Chain(8, TS('t'), TS('f'), TS('i'), TS('('), NS('P'), TS(')'), NS('B'), NS('Z')),

Rule::Chain(7, TS('t'), TS('f'), TS('i'), TS('('), NS('P'), TS(')'), NS('B')),

Rule::Chain(7, TS('t'), TS('f'), TS('i'), TS('('), TS(')'), NS('B'), NS('Z')),

Rule::Chain(6, TS('t'), TS('f'), TS('i'), TS('('), TS(')'), NS('B')),

Rule::Chain(9, TS('a'), TS('t'), TS('f'), TS('i'), TS('('), NS('P'), TS(')'), TS(';'), NS('Z')),

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

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

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

),

//B -> NrE; | rE;

Rule(NS('B'), GRB\_ERRORS\_START + 2, 2,// Функция обязана возвращать значение

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

Rule::Chain(5, TS('{'), TS('r'), NS('E'), TS(';'), TS('}'))

),

//N -> nti; | nti;N | iwE;N | iwE; | pE; | pE;N | cCAN | cCA

Rule(NS('N'), GRB\_ERRORS\_START + 3, 8,// Недопустимая инструкция в теле функции

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

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

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

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

Rule::Chain(4, TS('p'), NS('E'), TS(';'), NS('N')),

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

Rule::Chain(4, TS('c'), NS('C'), NS('A'), NS('N')),

Rule::Chain(3, TS('c'), NS('C'), NS('A'))

),

//E-> i | l | (E) | i(W) | iM | lM | (E)M | i(W)M Неверно составленное выражение

Rule(NS('E'), GRB\_ERRORS\_START + 4, 10,

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

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

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

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

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

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

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

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

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

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

),

//M-> vE | vEM Неверное арифметическое выражение

Rule(NS('M'), GRB\_ERRORS\_START + 5, 2,

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

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

),

//P-> ti | ti,P Неверный список параметров

Rule(NS('P'), GRB\_ERRORS\_START + 6, 2,

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

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

),

//W-> i | l | i,W | l,W Неверный список аргументов

Rule(NS('W'), GRB\_ERRORS\_START + 7, 4,

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

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

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

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

),

//A-> {D}Ge{D} | {D}G | {D}e{D} | {D} Неверно составленное условное выражение

Rule(NS('A'), GRB\_ERRORS\_START + 8, 4,

Rule::Chain(8, TS('{'), NS('D'), TS('}'), NS('G'), TS('e'), TS('{'), NS('D'), TS('}')),

Rule::Chain(4, TS('{'), NS('D'), TS('}'), NS('G')),

Rule::Chain(7, TS('{'), NS('D'), TS('}'), TS('e'), TS('{'), NS('D'), TS('}')),

Rule::Chain(3, TS('{'), NS('D'), TS('}'))

),

//D-> iwE;D | iwE; | pE;D | pE; В условном блке могут содержаться только операции присваивания и вывода

Rule(NS('D'), GRB\_ERRORS\_START + 9, 4,

Rule::Chain(5, TS('i'), TS('w'), NS('E'), TS(';'), NS('D')),

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

Rule::Chain(4, TS('p'), NS('E'), TS(';'), NS('D')),

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

),

//G-> ecC{D} | ecC{D}G Неверно составленные блоки else if

Rule(NS('G'), GRB\_ERRORS\_START + 10, 2,

Rule::Chain(7, TS('e'), TS('c'), NS('C'), TS('{'), NS('D'), TS('}'), NS('G')),

Rule::Chain(6, TS('e'), TS('c'), NS('C'), TS('{'), NS('D'), TS('}'))

),

//C -> (EuE) Условием может являться только сравнение двух выражений

Rule(NS('C'), GRB\_ERRORS\_START + 11, 1,

Rule::Chain(5, TS('('), NS('E'), TS('u'), NS('E'), TS(')'))

)

);

**Приложение Е**

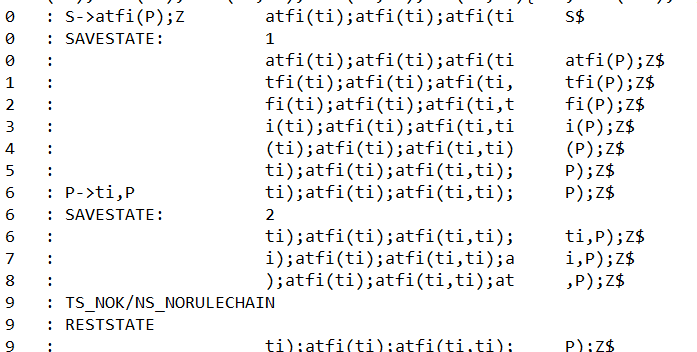


Рисунок Е.4 Начало синтаксического разбора

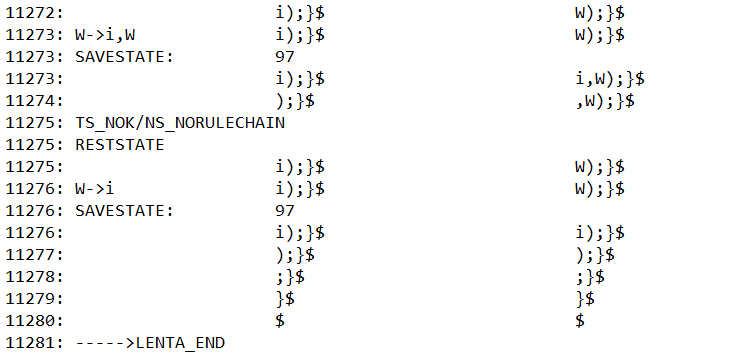


Рисунок Е.5 Окончание синтаксического разбора

**Приложение Ж**

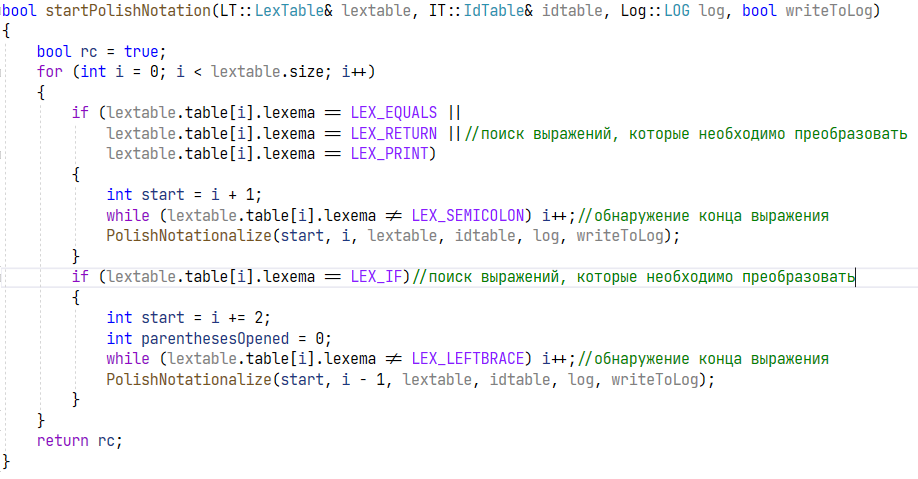


Рисунок Ж.6 Фрагмент кода транслятора, реализующего преобразование выражений в обратный польский формат, с краткими пояснениями

**Приложение З**

.586

.MODEL FLAT, STDCALL

INCLUDELIB libucrt.lib

INCLUDELIB kernel32.lib

INCLUDELIB "..\Debug\STDLIB.lib"

ExitProcess PROTO : DWORD

.STACK 4096

printerror PROTO : DWORD

printnum PROTO : SDWORD

stringcopy PROTO : DWORD, : DWORD

printstr PROTO : DWORD

strconcat PROTO : DWORD, : DWORD

stringcompare PROTO : DWORD, : DWORD

sumsymbol PROTO : DWORD, : DWORD

tonumber PROTO : DWORD

stringlength PROTO : DWORD

indexof PROTO : DWORD, : DWORD

userstringcompare PROTO : DWORD, : DWORD

fii PROTO : SDWORD, : SDWORD

fss PROTO : DWORD, : DWORD

.CONST

Ls000 BYTE 14, " is less than "

Ls001 BYTE 8, " equals "

Ls002 BYTE 16, " is bigger than "

Li003 SDWORD 1

Li004 SDWORD 5

Ls005 BYTE 10, "1234567890"

Ls006 BYTE 12, "123456789 0"

.DATA

fii@x SDWORD 0

fii@y SDWORD 0

fii@z SDWORD 0

fss@a BYTE 256 DUP(0)

fss@b BYTE 256 DUP(0)

fss@c BYTE 256 DUP(0)

main@x SDWORD 0

main@y SDWORD 0

main@sa BYTE 256 DUP(0)

main@sb BYTE 256 DUP(0)

main@sc BYTE 256 DUP(0)

.CODE

main PROC

;-------------l-------------

PUSH Li003

POP EAX

MOV main@x, EAX

;-------------l-------------

PUSH Li004

POP EAX

MOV main@y, EAX

;-------------l-------------

PUSH OFFSET Ls005

PUSH OFFSET main@sa

CALL stringcopy

;-------------l-------------

PUSH OFFSET Ls006

PUSH OFFSET main@sb

CALL stringcopy

;-------------ii@i-------------

PUSH main@y

PUSH main@x

CALL fii

PUSH EAX

CALL printnum

;-------------ii@i-------------

PUSH OFFSET main@sb

PUSH OFFSET main@sa

CALL fss

PUSH EAX

PUSH OFFSET main@sc

CALL stringcopy

;-------------i-------------

PUSH OFFSET main@sc

CALL printstr

;-------------i@i-------------

PUSH OFFSET main@sc

CALL stringlength

PUSH EAX

CALL printnum

;-------------ii@i-------------

PUSH OFFSET main@sb

PUSH OFFSET main@sa

CALL userstringcompare

PUSH EAX

CALL printnum

PUSH 0

CALL ExitProcess

main ENDP

;----------FUNCTION fii---------

fii PROC uses EBX ECX EDX, fii@x\_PARAM : SDWORD, fii@y\_PARAM : SDWORD

MOV EAX, fii@x\_PARAM

MOV fii@x, EAX

MOV EAX, fii@y\_PARAM

MOV fii@y, EAX

;-------------iiivv-------------

PUSH fii@x

PUSH fii@x

PUSH fii@y

POP EDX

POP EBX

ADD EBX, EDX

PUSH EAX

POP EDX

POP EBX

IMUL EBX, EDX

PUSH EBX

POP EAX

MOV fii@z, EAX

;-------------i-------------

PUSH fii@z

POP EAX

RET

fii ENDP

;----------FUNCTION fss---------

fss PROC uses EBX ECX EDX, fss@a\_OFFSET : DWORD, fss@b\_OFFSET : DWORD

PUSH fss@a\_OFFSET

PUSH OFFSET fss@a

CALL stringcopy

PUSH fss@b\_OFFSET

PUSH OFFSET fss@b

CALL stringcopy

;-------------iiu-------------

PUSH OFFSET fss@a

PUSH OFFSET fss@b

POP EBX

POP EAX

PUSH EBX

PUSH EAX

CALL stringcompare

MOV EBX, 0

.IF (EAX < EBX)

;-------------ilviv-------------

PUSH OFFSET fss@a

PUSH OFFSET Ls000

POP ECX

POP EBX

INVOKE strconcat, EBX, ECX

PUSH EAX

PUSH OFFSET fss@b

POP ECX

POP EBX

INVOKE strconcat, EBX, ECX

PUSH EAX

PUSH OFFSET fss@c

CALL stringcopy

;-------------ii@i-------------

PUSH OFFSET fss@b

PUSH OFFSET fss@a

CALL indexof

PUSH EAX

CALL printnum

.ELSE

;-------------iiu-------------

PUSH OFFSET fss@a

PUSH OFFSET fss@b

POP EBX

POP EAX

PUSH EBX

PUSH EAX

CALL stringcompare

MOV EBX, 0

.IF (EAX == EBX)

;-------------ilviv-------------

PUSH OFFSET fss@a

PUSH OFFSET Ls001

POP ECX

POP EBX

INVOKE strconcat, EBX, ECX

PUSH EAX

PUSH OFFSET fss@b

POP ECX

POP EBX

INVOKE strconcat, EBX, ECX

PUSH EAX

PUSH OFFSET fss@c

CALL stringcopy

;-------------i@i-------------

PUSH OFFSET fss@a

CALL stringlength

PUSH EAX

CALL printnum

JMP endIfMarker@88

.ENDIF

;-------------ilviv-------------

PUSH OFFSET fss@a

PUSH OFFSET Ls002

POP ECX

POP EBX

INVOKE strconcat, EBX, ECX

PUSH EAX

PUSH OFFSET fss@b

POP ECX

POP EBX

INVOKE strconcat, EBX, ECX

PUSH EAX

PUSH OFFSET fss@c

CALL stringcopy

;-------------i@i-------------

PUSH OFFSET fss@a

CALL tonumber

PUSH EAX

CALL printnum

endIfMarker@88:

.ENDIF

;-------------i-------------

PUSH OFFSET fss@c

POP EAX

RET

fss ENDP

END main