МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программная инженерия

Специальность 1-40 01 01 «Программное обеспечение информационных технологий»

Специализация 1-40 01 01 «Программное обеспечение информационных технологий (программирование интернет-приложений)»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора SNE-2023»

Выполнил студент Селицкий Николай Евгеньевич

(Ф.И.О.)

Руководитель проекта асс. Мущук Артур Николаевич

(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.)

Консультант асс. Мущук Артур Николаевич

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

Нормоконтролер асс. Мущук Артур Николаевич

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

Курсовой проект защищен с оценкой

Минск 2023

Оглавление

[Введение 4](#_Toc153806148)

[1 Спецификация языка программирования 5](#_Toc153806149)

[1.1 Характеристика языка программирования 5](#_Toc153806150)

[1.2 Определение алфавита языка программирования 5](#_Toc153806151)

[1.3 Применяемые сепараторы 5](#_Toc153806152)

[1.4 Применяемые кодировки 6](#_Toc153806153)

[1.5 Типы данных 6](#_Toc153806154)

[1.6 Преобразование типов данных 7](#_Toc153806155)

[1.7 Идентификаторы 7](#_Toc153806156)

[1.8 Литералы 7](#_Toc153806157)

[1.9 Объявление данных 8](#_Toc153806158)

[1.10 Инициализация данных 9](#_Toc153806159)

[1.11 Инструкции языка 9](#_Toc153806160)

[1.12 Операции языка 9](#_Toc153806161)

[1.13 Выражения и их вычисление 10](#_Toc153806162)

[1.14 Конструкции языка 10](#_Toc153806163)

[1.15 Области видимости идентификаторов 11](#_Toc153806164)

[1.16 Семантические проверки 11](#_Toc153806165)

[1.17 Распределение оперативной памяти на этапе выполнения 12](#_Toc153806166)

[1.18 Стандартная библиотека и её состав 12](#_Toc153806167)

[1.19 Ввод и вывод данных 12](#_Toc153806168)

[1.20 Точка входа 13](#_Toc153806169)

[1.21 Препроцессор 13](#_Toc153806170)

[1.22 Соглашения о вызовах 13](#_Toc153806171)

[1.23Обьектный код 13](#_Toc153806172)

[1.24 Классификация сообщений транслятора 13](#_Toc153806173)

[1.25 Контрольный пример 13](#_Toc153806174)

[2 Структура транслятора 15](#_Toc153806175)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 15](#_Toc153806176)

[2.2 Перечень параметров транслятора 16](#_Toc153806177)

[2.3 Протоколы, формируемые транслятором 16](#_Toc153806178)

[3 Разработка лексического анализатора 18](#_Toc153806179)

[3.1 Структура лексического анализатора 18](#_Toc153806180)

[3.2. Входные и выходные данные лексического анализатора 19](#_Toc153806181)

[3.3 Параметры лексического анализатора 19](#_Toc153806182)

[3.4 Алгоритм лексического анализа 19](#_Toc153806183)

[3.5 Контроль входных символов 19](#_Toc153806184)

[3.6 Удаление избыточных символов 20](#_Toc153806185)

[3.7 Перечень ключевых слов 20](#_Toc153806186)

[3.8 Основные структуры данных 22](#_Toc153806187)

[3.9 Структура и перечень сообщений лексического анализатора 23](#_Toc153806188)

[3.10 Принцип обработки ошибок 23](#_Toc153806189)

[3.11 Контрольный пример 23](#_Toc153806190)

[4. Разработка синтаксического анализатора 24](#_Toc153806191)

[4.1 Структура синтаксического анализатора 24](#_Toc153806192)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 24](#_Toc153806193)

[4.3 Построение конечного магазинного автомата 26](#_Toc153806194)

[4.4 Основные структуры данных 27](#_Toc153806195)

[4.5 Описание алгоритма синтаксического разбора 27](#_Toc153806196)

[4.6 Параметры синтаксического анализатора и режимы его работы 27](#_Toc153806197)

[4.7 Структура и перечень сообщений синтаксического анализатора 27](#_Toc153806198)

[4.8. Принцип обработки ошибок 28](#_Toc153806199)

[4.9. Контрольный пример 28](#_Toc153806200)

[5 Разработка семантического анализатора 28](#_Toc153806201)

[5.1 Структура семантического анализатора 28](#_Toc153806202)

[5.2 Функции семантического анализатора 28](#_Toc153806203)

[5.3 Структура и перечень сообщений семантического анализатора 29](#_Toc153806204)

[5.4 Принцип обработки ошибок 29](#_Toc153806205)

[5.5 Контрольный пример 29](#_Toc153806206)

[6. Вычисление выражений 30](#_Toc153806207)

[6.1 Выражения, допускаемые языком 31](#_Toc153806208)

[6.2 Польская запись и принцип её построения 31](#_Toc153806209)

[6.3 Программная реализация обработки выражений 31](#_Toc153806210)

[6.4 Контрольный пример 32](#_Toc153806211)

[7. Генерация кода 33](#_Toc153806212)

[7.1 Структура генератора кода 33](#_Toc153806213)

[7.2 Представление типов данных в оперативной памяти 33](#_Toc153806214)

[7.3 Статическая библиотека 33](#_Toc153806215)

[7.4 Особенности алгоритма генерации кода 34](#_Toc153806216)

[7.5 Параметры, управляющие генерацией кода 34](#_Toc153806217)

[7.6 Контрольный пример 34](#_Toc153806218)

[8. Тестирование транслятора 36](#_Toc153806219)

[8.1 Общие приложения 36](#_Toc153806220)

[8.2 Результаты тестирования 36](#_Toc153806221)

[Заключение 39](#_Toc153806222)

[Графический материал 40](#_Toc153806223)

[Список использованных источников 41](#_Toc153806224)

[Приложение А 42](#_Toc153806225)

[Приложение Б 44](#_Toc153806226)

[Приложение Г 50](#_Toc153806227)

[Приложение Д 56](#_Toc153806228)

# Введение

В данном курсовом проекте поставлена задача разработки собственного языка программирования и транслятора для него. Название языка – SNE-2023. Написание транслятора будет осуществляться на языке C++, при этом код на языке SNE-2023 будет транслироваться в язык ассемблера. Курсовой проект выполнялся с целью глубокого изучения и понимания работы компилятора.

Задание на курсовой проект можно разделить на следующие задачи:

1. Разработка спецификации языка SNE-2023;
2. Разработка лексического анализатора;
3. Разработка синтаксического анализатора;
4. Разработка семантического анализатора;
5. Разбор арифметических выражений;
6. Разработка генератора кода;
7. Тестирование транслятора.

# 1 Спецификация языка программирования

## 1.1 Характеристика языка программирования

Язык программирования SNE-2023 является процедурным, универсальным, строго типизированным, не объектно-ориентированным, компилируемым.

## 1.2 Определение алфавита языка программирования

При написании программы на языке SNE-2023 используется таблица символов ASCII, представленная в рис.1.1.

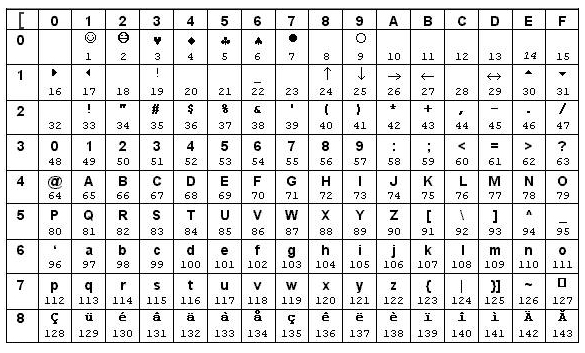


Рисунок 1.1 Алфавит входных символов

Символы, используемые на этапе выполнения: [a…z], [A…Z], [0…9], символы пробела, табуляции и перевода строки, спецсимволы: ( ) , ; + - / \* % > < = ! ?

## 1.3 Применяемые сепараторы

Символы-сепараторы служат в качестве разделителей цепочек языка во время обработки исходного текста программы с целью разделения на токены. Они представлены в таблице 1.2.

Таблица 1.2 Символы-сепараторы

|  |  |
| --- | --- |
| Символ(ы) | Назначение |
| ‘пробел’, \t | Разделитель цепочек. Допускается везде кроме названий идентификаторов и ключевых слов |

Продолжение таблицы 1.2

|  |  |
| --- | --- |
| Символ(ы) | Назначение |
| **(** … **)** | Блок параметров функции и приоритет арифметических операций |
| **,** | Разделитель параметров функций |
| **?** | Символ, отделяющий условные конструкции |
| **+ - \* /** % | Арифметические операции |
| **> <** = | Логические операции (операции сравнения: больше, меньше, равно), используемые в условной конструкции. |
| ; | Разделитель программных конструкций |
| }{ | Операторы разделяющие области видимости |
| **=** | Оператор присваивания |
| **\**n | Переход и создание новой строки |

## 1.4 Применяемые кодировки

Для написания программ язык SNE-2023 использует кодировку ASCII, содержащую английский алфавит, а также некоторые специальные символы, такие как ( ) , ; + - / \* % > < = ! ?.

## 1.5 Типы данных

В языке SNE-2023 реализованы два фундаментальных типа данных: целочисленный и строковый. Описание типов приведено в таблице 1.3.

Таблица 1.3 Типы данных языка SNE-2023

|  |  |
| --- | --- |
| Тип данных | Характеристика |
| Целочисленный тип данных **number** | Фундаментальный тип данных. Используется для работы с числовыми значениями. В памяти занимает 2 байта. Максимальное значение: 32767.  Минимальное значение: -32768.  Инициализация по умолчанию: значение 0.  Поддерживаемые операции:  **+** (бинарный) – оператор сложения;  - (бинарный) – оператор вычитания;  - (унарный) –делает запись типа ‘0 минус число’;  **\*** (бинарный) – оператор умножения;  **/** (бинарный) – оператор деления нацело;  % (бинарный) – оператор остатка от деления;  **=** (бинарный) – оператор присваивания.  В качестве оператора условия поддерживаются операции:  **>** (бинарный) – оператор «больше»;  **<** (бинарный) – оператор «меньше»;  = (бинарный) – оператор проверки на равенство. |

Продолжение таблицы 1.3

|  |  |
| --- | --- |
| Тип данных | Характеристика |
| Строковый тип данных **string** | Фундаментальный тип данных. Используется для работы с символами, каждый из которых занимает 1 байт. Максимальное количество символов – 256.  Инициализация по умолчанию: строка нулевой длины “”. Операции над данными строкового типа: присваивание строковому идентификатору значения другого строкового идентификатора, строкового литерала или значения строковой функции, а также использование библиотечных функций(метод переводящий строку в число). |

## 1.6 Преобразование типов данных

Преобразование типов данных в языке SNE-2023 возможно в явном виде благодаря встроенному методу ‘texttolit’, который переводит строку в число.

## 1.7 Идентификаторы

Общее количество идентификаторов ограничено максимальным размером таблицы идентификаторов. Идентификаторы должны содержать только символы нижнего регистра латинского алфавита. Максимальная длина идентификатора равна 10 символам. Идентификаторы, объявленные внутри функционального блока, получают массив вложенности, идентичный вложенности функции, внутри которой они объявлены. Массив состоит из 5 элементов что означает что максимальное количество вложенностей не может превышать 5. В случае превышения заданной границы выводиться ошибка. Имена идентификаторов усекаются до длины, равной 15 символов. Данные правила действуют для всех типов идентификаторов. Зарезервированный идентификатор ‘buffer’. Идентификаторы не должны совпадать с ключевыми словами. Типы идентификаторов: имя переменной, тип, параметр функции. Имена идентификаторов-функций не должны совпадать с именами команд ассемблера (это не касается имён идентификаторов-переменных).

Примеры идентификаторов

|  |  |
| --- | --- |
| Правильные | Неправильные |
| hello  first  f | Hello  Первый  5 |

## 1.8 Литералы

С помощью литералов осуществляется инициализация переменных.

Все литералы делятся на vint, vstr. Типы литералов языка SNE-2023 представлены в таблице 1.4.

Таблица 1.4 Литералы

|  |  |
| --- | --- |
| Литералы | Пояснение |
| Целочисленные литералы в десятичном представлении | Последовательность цифр 0…9 с предшествующим знаком минус или без него (знак минус не должен отделяться пробелом) |
| Строковые литералы | Набор символов (от 1 до 255), заключённых в одиночных кавычки |
| Целочисленные литералы в восмеричном представлении | Последовательность цифр 0…7 с символом ‘!’ в начале |
| Примеры литералов | |
| Правильные | Неправильные |
| ‘hello’  72  !13 | ‘hello  7g2  !9 |

Ограничения на строковые литералы языка SNE-2023: внутри литерала не допускается использование символов кириллицы, а также одинарных и двойных кавычек. Ограничения на целочисленные литералы: не могут начинаться с нуля, если их значение не ноль; если литерал отрицательный, после знака “-” не может быть нуля.

## 1.9 Объявление данных

Для объявления переменной используется ключевое слово **new**, после которого указывается тип данных и имя идентификатора. Не допускается инициализация при объявлении.

Таблица 1.5 пример создания переменных

|  |  |
| --- | --- |
| Пример объявления числового типа | Пример объявления переменной строкового типа |
| new little numb; | new text str; |

Для объявления функций используется ключевое слово **function**, перед которым указывается тип функции и после которого имя функции. Далее обязателен список параметров и тело функции.

Для отслеживания области видимости каждый идентификатор имеет массив вложенности из пяти символов в каждый из которых записываются определенные числа соответствующие номерам и последовательности блоков в которые этот идентификатор был вложен.

## 1.10 Инициализация данных

При объявлении переменной не допускается инициализация данных. Объектами-инициализаторами могут быть только идентификаторы или литералы. При объявлении без инициализации предусмотрены значения по умолчанию: значение 0 для типа **little** и строка длины 0 (‘’) для типа **text**.

Пример

|  |  |
| --- | --- |
| Пример инициализации данных | Значение по умолчанию |
| New text h; | h = ‘’ - по умолчанию |
| New little n; | n = 0 - по умолчанию |

## 1.11 Инструкции языка

Инструкции языка SNE-2023 представлены в таблице 1.6.

Таблица 1.6 ИнструкцииязыкаSNE-2023

|  |  |
| --- | --- |
| Инструкция | Реализация |
| Объявление переменной | new <тип данных> <идентификатор>; |
| Возврат из функции | Все функции в языке SNE-2023 возвращают какое либо значение:  return <идентификатор/литерал>; |
| Вывод данных | output<идентификатор/литерал>; |
| Вызов функции | <Тип функции> function <идентификатор функции> (<список параметров>){…};  Список параметров может быть пустым. |
| Присваивание | <идентификатор> = <выражение>;  Выражением может быть идентификатор, литерал, или вызов функции. Для целочисленного типа выражение может быть дополнено арифметическими операциями с любым количеством операндов с использованием скобок. Для строкового типа выражение может быть только идентификатором, литералом или вызовом функции, возвращающей значение строкового типа. |

## 1.12 Операции языка

В языке SNE-2023 предусмотрены следующие операции с данными. Приоритетность операции умножения выше приоритета операций сложения и вычитания. Для установки наивысшего приоритета используются круглые скобки. Операции языка представлены в таблице 1.7.

Таблица 1.7 Операции языка SNE-2023

|  |  |
| --- | --- |
| Тип оператора | Оператор |
| Арифметические | 1. **+ –** сложение (коммутативно) 2. - – вычитание 3. \* –умножение(ассоциативно, коммутативно, дистрибутивно) 4. **/** – деление нацело 5. % – остаток от деления 6. = – присваивание |
| Строковые | 1. **=** – присваивание |
| Логические | 1. **>** – больше  2. **<** – меньше  3. = – равенство |

## 1.13 Выражения и их вычисление

Вычисление выражений – одна из важнейших задач языков программирования. Всякое выражение составляется согласно следующим правилам:

1. Допускается использовать скобки для смены приоритета операций;
2. Выражение записывается в строку без переносов;
3. Использование двух подряд идущих операторов не допускается;

Перед генерацией кода каждое выражение приводится к записи в польской записи для удобства дальнейшего вычисления выражения на языке ассемблера.

## 1.14 Конструкции языка

Программа на языке SNE-2023 оформляется в виде функций пользователя и главной функции. При составлении функций рекомендуется выделять блоки и фрагменты и применять отступы для лучшей читаемости кода.

Программные конструкции языка представлены в таблице 1.8.

Таблица 1.8 Программные конструкции языка SNE-2023

|  |  |
| --- | --- |
| Конструкция | Реализация |
| Главная функция | **start**  **{**  …  **}** |
| Внешняя функция | <тип данных> **function** <идентификатор> **(**<тип> <идентификатор>, ...**)**  **{**  …  **return** <идентификатор/литерал>;  **}** |
| Условная конструкция | **check(**<идентификатор1> <оператор> <идентификатор2>**)?**  **{…}**  **not**  **{…}**  <идентификатор1>, <идентификатор2> - идентификаторы или литералы целочисленного типа. <оператор> - один из операторов сравнения ( **> < =** ), устанавливающий отношение между двумя операндами и организующий условие данной конструкции. При истинности условия выполняется код внутри блока **check**, иначе – код внутри блока **not**. Блок **not** не обязательный. Обращаю внимание на то, что в конце описания строки с **check** долен стоять **‘?’**. |

## 

## 1.15 Области видимости идентификаторов

Область видимости: сверху вниз. Переменные, объявленные в одной функции, недоступны в другой. Каждая переменная или параметр функции получают свой массив вложенности, который указывает в каком блоке они находиться.

Есть глобальная переменная buffer, которая не требует инициализации, а также есть возможность создания собственных глобальных переменных. Параметры объявленные внутри функций видны только внутри функции, в которой объявлены.

## 1.16 Семантические проверки

В языке программирования SNE-2023 выполняются следующие семантические проверки:

1. Наличие функции **start** – точки входа в программу;
2. Единственность точки входа;
3. Неправильные вычисления (например деление на 0)
4. Подсчет количество параметров специальных методов
5. Проверка типа параметров специального метода
6. Подсчет количества операндов в выражении (не> 8)
7. Проверка типа возвращаемого значение
8. Проверка что метод возвращает значение (все методы включая start должны возвращать значение)
9. Проверка максимальное количество передаваемых параметров при определении функции
10. Проверка максимальное количество передаваемых параметров при вызове функции
11. Проверка типа параметров
12. Проверка количества параметров

## 1.17 Распределение оперативной памяти на этапе выполнения

Транслированный код использует две области памяти: сегмент констант и сегмент данных. В сегменте констант хранятся все литералы, такие как числа и строки. В сегменте данных хранятся переменные и параметры функций.

Локальная область видимости в исходном коде определяется с помощью правил именования идентификаторов и ограничивается массивом вложенности. В исходном коде переменные и параметры функций могут быть видимы только в определенных блоках кода, определенных с помощью фигурных скобок.

## 1.18 Стандартная библиотека и её состав

В языке SNE-2023 присутствует стандартная библиотека, которая подключается автоматически при трансляции исходного кода в язык ассемблера. Содержимое библиотеки и описание функций представлено в таблице 1.9.

Таблица 1.9 Стандартная библиотека языка SNE-2023

|  |  |
| --- | --- |
| Функция | Описание |
| **int** texttolit(**char**\* str1) | Целочисленная функция. Преобразует строку в число |
| **int** textlenght(**char**\* str1) | Целочисленная функция. Вычисляет и возвращает длину строки str |

Стандартная библиотека написана на языке С, подключается к транслированному коду на этапе генерации кода. Вызовы стандартных функций доступны там же, где и вызов пользовательских функций. Также в стандартной библиотеке реализованы функции для манипулирования выводом, недоступные конечному пользователю. Для вывода предусмотрен оператор **output**. Эти функции представлены в таблице 1.10.

Таблица 1.10 Дополнительные функции стандартной библиотеки

|  |  |
| --- | --- |
| Функция на языке С++ | Описание |
| **char**\*copytxt(**char**\* str1, **char**\* str2) | Функция для копирования строки из str1 в str2 помогающая осуществить генерацию |
| **int** outtxt(**char**\* value) | Функция для вывода в стандартный поток текста |
| **int** outlit(**int** value) | Функция для вывода в стандартный поток числа |

## 1.19 Ввод и вывод данных

В языке SNE-2023 не организован ввод данных с консоли.

Вывод данных осуществляется с помощью оператора **output**. Допускается использование оператора **output** с литералами и идентификаторами.

Функции, управляющие выводом данных, реализованы на языке С++ и вызываются из транслированного кода, конечному пользователю недоступны. Пользовательская команда **output** в транслированном коде будут заменена вызовом нужных библиотечных функций. Библиотека, содержащая нужные методы, подключается на этапе генерации кода.

## 1.20 Точка входа

В языке SNE-2023 каждая программа должна содержать главную функцию (точку входа) **start**, с первой инструкции которой начнётся последовательное выполнение команд программы.

## 1.21 Препроцессор

Препроцессор, принимающий и выдающий некоторые данные на вход транслятору, в языке SNE-2023 отсутствует.

## 1.22 Соглашения о вызовах

В языке вызов функций происходит по соглашению о вызовах stdcall. Осо-бенности stdcall:

– все параметры функции передаются через стек;

– память высвобождает вызываемый код;

– занесение в стек параметров идёт справа налево.

**1.23Обьектный код**

После выполнения генерации создается файл с сгенерированным кодом на языке ассемблера.

## 1.24 Классификация сообщений транслятора

Сообщения транслятора приведены в таблице 1.11, а также в приложении А.

Таблица 1.11 Классификация ошибок

|  |  |
| --- | --- |
| Номера ошибок | Характеристика |
| 0 – 99 | Ошибки системные |
| 100 – 109 | Ошибки параметров |
| 110 – 129 | Ошибки лексического анализа |
| 130 – 149 | Ошибки синтаксического анализа |
| 150 – 179 | Ошибки семантического анализа |
| 180 – 200 | Ошибки другие |

## 1.25 Контрольный пример

Контрольный пример демонстрирует главные особенности языка SNE-2023: его фундаментальные типы, основные структуры, функции, использование функций статической библиотеки. Исходный код контрольного примера представлен в приложении А. Сам контрольный пример вместе с другими примерами храниться в файле **input2.txt**.

# 2 Структура транслятора

## 2.1 Компоненты транслятора, их назначение и принципы взаимодействия

В языке SNE-2023 исходный код транслируется в язык Assembler. Транслятор языка разделён на отдельные части, которые взаимодействуют между собой и выполняют отведённые им функции, которые представлены в пункте 2.1. Для того чтобы получить ассемблерный код, используется выходные данные работы лексического анализатора, а именно таблица лексем и таблица идентификаторов. Для указания выходных файлов используются входные параметры транслятора, которые описаны в таблице 2.1.

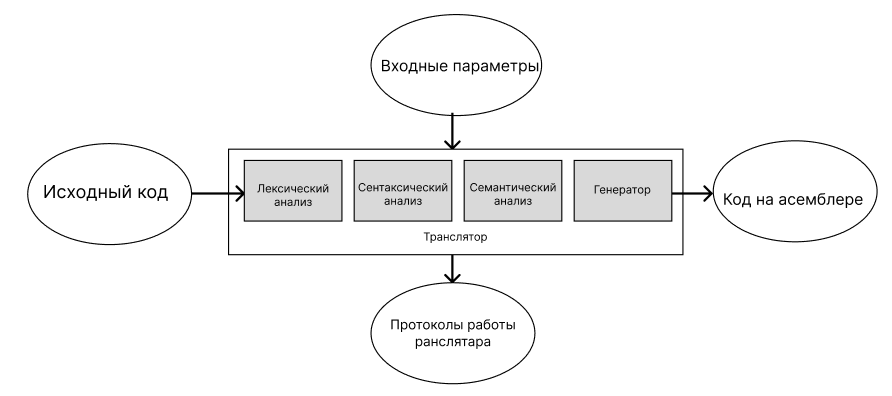


Рисунок 2.1 Структура транслятора языка программирования SNE-2023

Первая стадия работы компилятора называется лексическим анализом, а программа, её реализующая, – лексическим анализатором (сканером). На вход лексического анализатора подаётся последовательность символов входного языка. Сканер производит предварительный разбор текста, преобразующий единый массив текстовых символов в массив отдельных слов (в теории компиляции вместо термина «слово» часто используют термин «токен»). Примеры лексических единиц: идентификаторы, числа, символы операций, служебные слова и т.д. Лексический анализатор преобразует исходный текст, заменяя лексические единицы их внутренним представлением – лексемами, для создания промежуточного представления исходной программы. Каждой лексеме сопоставляется ее тип и запись в таблице идентификаторов, в которой хранится дополнительная информация. Таблица лексем (ТЛ) и таблица идентификаторов (ТИ) являются входом для следующей фазы компилятора – синтаксического анализа (разбора, парсера).

Цели лексического анализатора:

− убрать все лишние пробелы;

− выполнить распознавание лексем;

− построить таблицу лексем и таблицу идентификаторов;

− при неуспешном распознавании или обнаружении некоторых ошибок во входном тексте выдать сообщение об ошибке.

Синтаксический анализатор – часть компилятора, выполняющая синтаксический анализ, то есть проверку исходного кода на соответствие правилам грамматики. Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов. Выходной информацией является дерево разбора.

Семантический анализатор – часть транслятора, выполняющая семантический анализ, то есть проверку исходного кода на наличие ошибок, которые невозможно отследить при помощи регулярной и контекстно-свободной грамматики. Входными данными являются таблица лексем и идентификаторов.

Генератор кода – часть транслятора, выполняющая генерацию ассемблерного кода на основе полученных данных на предыдущих этапах трансляции. На вход генератора подаются таблица лексем и таблица идентификаторов, на основе которых генерируется файл с ассемблерным кодом.

## 2.2 Перечень параметров транслятора

Для формирования файлов с результатами работы лексического, синтаксического и семантического анализаторов используются входные параметры транслятора, которые приведены в таблице 2.2.

Таблица 2.2 Входные параметры транслятора языка SNE-2023

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание параметра | Значение по умолчанию |
| -in:<путь к in-файлу> | Файл с исходным кодом на языке SNE-2023 , имеющий расширение .txt | Не предусмотрено |
| -log:<путь к log-файлу> | Файл журнала для вывода протоколов работы программы. Расщирение .log | Значение по умолчанию:  input.log |

## 2.3 Протоколы, формируемые транслятором

Протоколы, формируемые транслятором, представлены в таблице 2.3.

Таблица 2.3 Протоколы, формируемые транслятором языка SNE-2023

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |
| Файл журнала, заданный параметром "-log:" | Файл с протоколом работы транслятора языка программирования SNE-2023 . Содержит таблицу лексем(в двух видах) и таблицу идентификаторови дерево разбора, полученные на этапе лексического и синтаксического анализа. |

Таблица 2.3 продолжение

|  |  |
| --- | --- |
| Файл журнала, trac.txt | Файл протокол хранящий визуальное отображение работы синтаксического анализатора и хранящий ошибки возникшие при его работе. |
| Выходной файл, c расширением ".asm" | Результат работы программы – файл, содержащий исходный код на языке ассемблера. |

В ходе работы программы формируются протоколы работы лексического, синтаксического и семантического анализаторов, которые содержат в себе перечень протоколов работы. В таблице 2.3 приведены протоколы, формируемые транслятором и их содержимое.

# 3 Разработка лексического анализатора

## 3.1 Структура лексического анализатора

Первая стадия работы компилятора называется лексическим анализом, а программа, её реализующая, – лексическим анализатором (сканером). На вход лексического анализатора подаётся исходный код входного языка. Лексический анализатор выделяет в этой последовательности простейшие конструкции языка,. Лексический анализатор производит предварительный разбор текста, преобразующий единый массив текстовых символов в массив токенов.

Примеры лексических единиц: идентификаторы, числа, символы операций, служебные слова и т.д. Лексический анализатор преобразует исходный текст, заменяя лексические единицы их внутренним представлением – лексемами, для создания промежуточного представления исходной программы. Каждой лексеме сопоставляется ее тип и запись в таблице идентификаторов, в которой хранится дополнительная информация.

Функции лексического анализатора:

− удаление «пустых» символов и комментариев. Если «пустые» символы (пробелы, знаки табуляции и перехода на новую строку) и комментарии будут удалены лексическим анализатором, синтаксический анализатор никогда не столкнется с ними;

− распознавание идентификаторов и ключевых слов;

− распознавание констант;

− распознавание разделителей и знаков операций.

− определение вложенности.

Структура лексического анализатора представлена на рисунке 3.1.

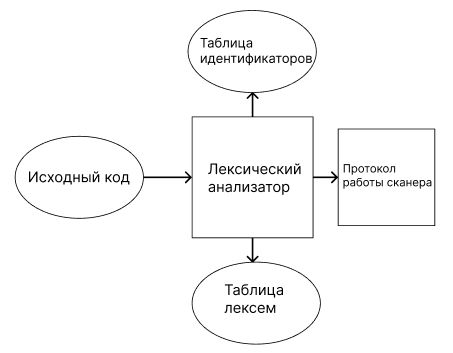


Рисунок 3.1 Структура лексического анализатора

## 

## 3.2. Входные и выходные данные лексического анализатора

Входные данные код прошедший проверку, выходные таблица лексем и таблица идентификаторов.

## 3.3 Параметры лексического анализатора

Результаты работы лексического анализатора, а именно таблицы лексем и идентификаторов выводятся как в файл журнала, так и в отдельные файлы.

## 3.4 Алгоритм лексического анализа

* проверяет входной поток символов программы на исходном языке на допустимость, удаляет лишние пробелы и добавляет сепаратор для вычисления номера строки для каждой лексемы;
* для выделенной части входного потока выполняется функция распознавания лексемы;
* при успешном распознавании информация о выделенной лексеме заносится в таблицу лексем и таблицу идентификаторов, и алгоритм возвращается к первому этапу;
* формирует протокол работы;
* при неуспешном распознавании выдается сообщение об ошибке.

Распознавание цепочек основывается на работе конечных автоматов. Работу конечного автомата можно проиллюстрировать с помощью графа переходов. Пример графа для цепочки «**little**» представлен на рисунке 3.2, где S0 – начальное, а S6 – конечное состояние автомата.

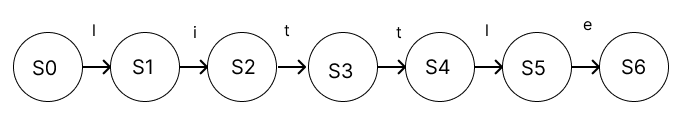


Рисунок 3.2 Пример графа переходов для цепочки string

## 3.5 Контроль входных символов

Для удобной работы с исходным кодом, при передаче его в лексический анализатор, все символы разделяются по категориям. Таблица входных символов представлена на рисунке 3.3, категории входных символов представлены в таблице 3.4.

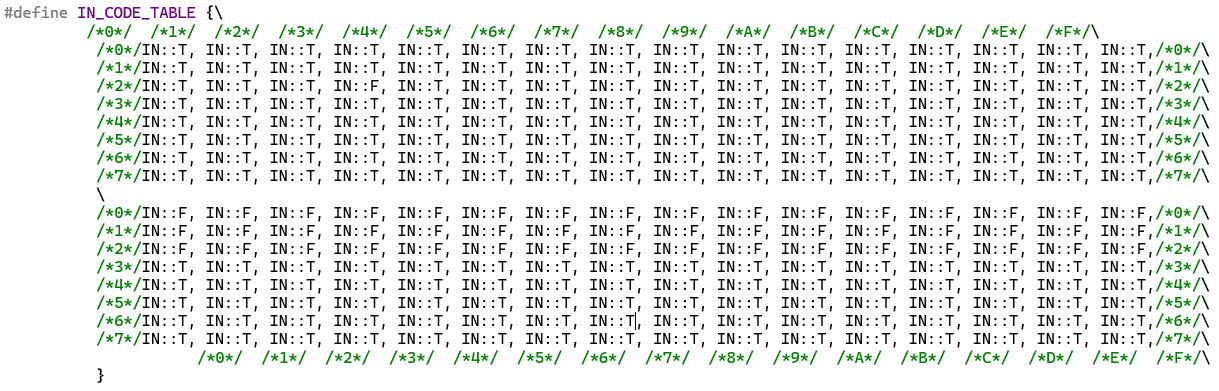


Рисунок 3.3. Таблица контроля входных символов

Таблица 3.4 Соответствие символов и их значений в таблице

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

## 3.6 Удаление избыточных символов

Избыточными символами являются символы табуляции и пробелы.

Избыточные символы удаляются на этапе разбиения исходного кода на токены.

Описание алгоритма удаления избыточных символов:

1. Посимвольно считываем файл с исходным кодом программы;

2. Встреча пробела или знака табуляции обрабатывается как встреча сепаратора;

3. В отличие от других символов-сепараторов не записываем в очередь лексем эти символы.

## 3.7 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 3.5.

Таблица 3.5 Соответствие токенов и сепараторов с лексемами

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Пояснение |
| little, text | t | Возможные типы |
| Идентификатор | i | Имя идентификатора |
| Литерал | l | Строковый или числовой |
| function | f | Создание функции |

Таблица 3.5 продолжение

|  |  |  |
| --- | --- | --- |
| new | n | Создание переменной |
| return | r | Возвращает значение из функции |
| output | o | Вывод в консоль |
| start | s | Главная функция |
| check | c | Конструкция условия |
| not | j | Необязательная вторая часть условия |
| ? | ? | Конец задачи условия |
| Встроенные функции | z | Функции, описанные в статической библиотеке |
| + - \* / % | v | Арифметические операторы |
| > < | y | Больше и меньше |
| = | = | Может быть как присвоение и как условие поэтому выносим отдельно |

Каждому выражению соответствует детерминированный конечный автомат, по которому происходит разбор данного выражения. На каждый автомат в массиве подаётся токен и с помощью регулярного выражения, соответствующего данному графу переходов, происходит разбор. В случае успешного разбора выражения оно записывается в таблицу лексем. Если выражение является идентификатором или литералом, информация также заносится в таблицу идентификаторов. Структура конечного автомата и пример графа перехода конечного автомата изображены на рисунках 3.6 и 3.7 соответственно.

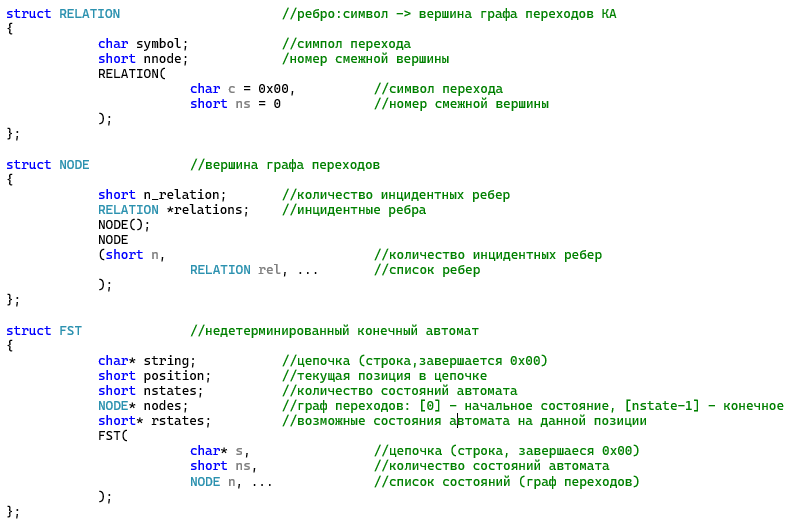
****

Рисунок 3.6 Структура конечного автомата

Далее приведен пример реализации графа конечного автомата. В качестве примера было взято спец слово function

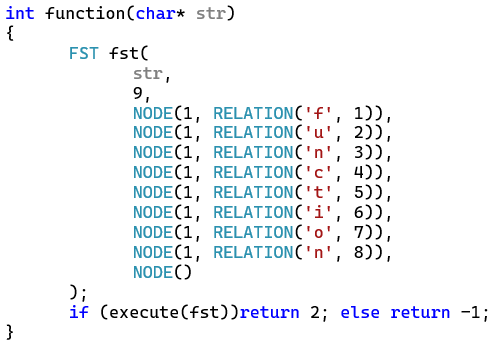


Рисунок 3.7 Пример реализации графа конечного автомата для токена function

## 3.8 Основные структуры данных

Основными структурами данных лексического анализатора являются таблица лексем и таблица идентификаторов. Таблица лексем содержит номер лексемы, строку на которой лексема находиться, лексему, номер идентификатора(если это идентификатор), значение(если это литерал или ? = l - целочисленный, t - строковый, ?-конец условия). Таблица идентификаторов содержит имя идентификатора (id), номер в таблице лексем (idxfirstLE), тип данных (iddatatype), тип идентификатора (idtype), значение (или параметры функций) (value) и массив вложенности. Код C++ со структурой таблицы лексем представлен на рисунке 3.8. Код C++ со структурой таблицы идентификаторов представлен на рисунке 3.9.

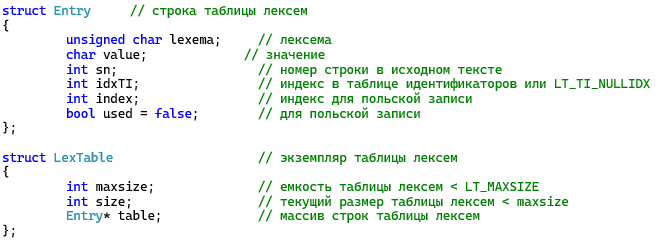


Рисунок 3.8 Структура таблицы лексем

На данных фото можно наглядно ознакомиться со структурами таблиц

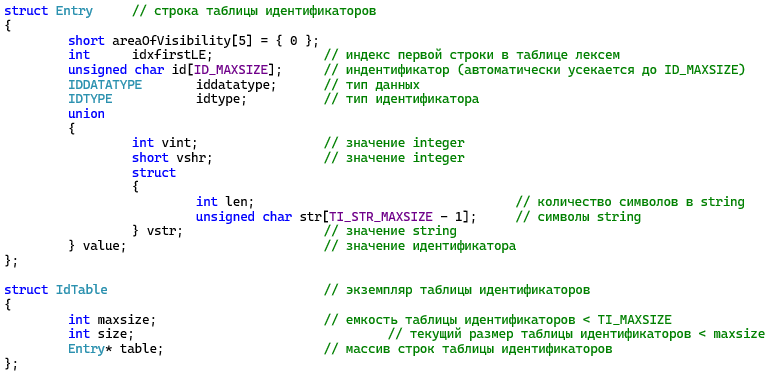


Рисунок 3.9 Структура таблицы идентификаторов

## 3.9 Структура и перечень сообщений лексического анализатора

Для обработки ошибок лексический анализатор использует таблицу с сообщениями. Структура сообщений содержит информацию о номере сообщения, номер строки и позицию, где было вызвано сообщение в исходном коде, информацию об ошибке. При возникновении сообщения, лексический анализатор игнорирует найденную ошибку и продолжает работу с исходным кодом. Перечень

сообщений представлен на рисунке 3.10.

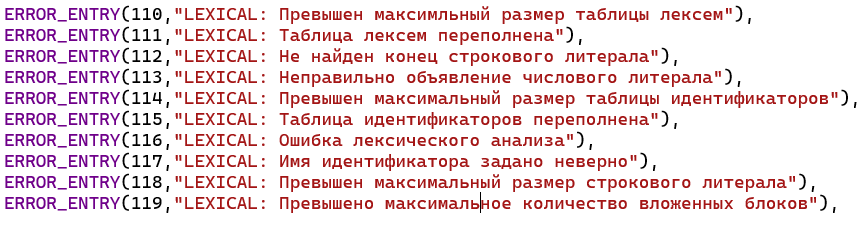


Рисунок 3.10 Сообщения лексического анализатора

## 3.10 Принцип обработки ошибок

При обнаружении ошибки лексер прерывает процесс компиляции и записывает в протокол ошибку с номером строки, где она была встречена. Программа перестает работать после первой найденной ошибки.

## 3.11 Контрольный пример

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

# 4. Разработка синтаксического анализатора

## 4.1 Структура синтаксического анализатора

Синтаксический анализатор: часть компилятора, выполняющая синтаксический анализ, то есть исходный код проверяется на соответствие правилам грамматики. Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов. Выходной информацией является дерево разбора.

Описание структуры синтаксического анализатора языка представлено на рисунке 4.1.

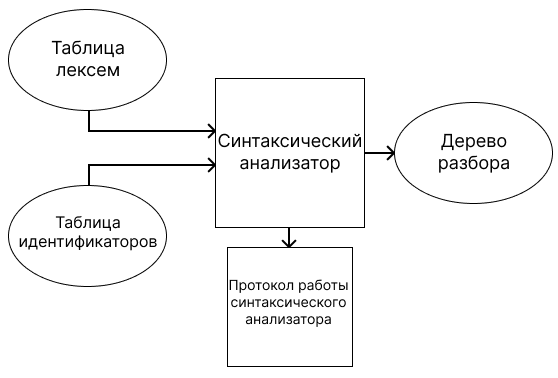


Рисунок 4.1 Структура синтаксического анализатора.

## 4.2 Контекстно-свободная грамматика, описывающая синтаксис языка

В синтаксическом анализаторе транслятора языка SNE-2023 используется контекстно-свободная грамматика , где

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

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

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

S – начальный символ грамматики, являющийся нетерминалом.

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

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

Описание нетерминальных символов содержится в таблице 4.2.

Таблица 4.2 Таблица правил переходов нетерминальных символов

|  |  |  |
| --- | --- | --- |
| Символ | Правила | Какие правила порождает |
| S | S->s{N};  S->s{N};S  S->tfi(){N};  S->tfi(F){N};  S->tfi(){N};S  S->tfi(F){N};S  S->nti;S  S->nti; | Стартовые правила, описывающее общую структуру программы |
| N | N->{N}N  N->{N}  N->nti;  N->nit;N  N->rU;  N->rU;N  N->i=E;  N->i=E;N  N->c(UCU)?{N}N  N->c(UCU)?{N}j{N}N  N->c(UCU)?{N}j{N}  N->c(UCU)?{N}  N->ntfi(F);  N->ntfi(F);N  N->o(U);N  N->o(U); | Правила для блоков кода в функциях |
| E | E->i  E->l  E->(E)  E->i(W)  E->i()  E->iM  E->lM  E->(E)M  E->i(W)M  E->i()M | Правила для идентификаторов |

Таблица 4.2 Продолжение

|  |  |  |
| --- | --- | --- |
| Символ | Правила | Какие правила порождает |
| F | F->ti  F->ti,F | Правила для параметров |
| W | W->i  W->l  W->i,W  W->l,W | Правила для идентификаторов и литералов |
| M | M->vE  M->vEM | Правила числа со знаком |
| U | U->i  U->l | Правила для переменных внутри check |
| C | C->=  C->y | Правила для знаков < > = |

## 4.3 Построение конечного магазинного автомата

Конечный автомат с магазинной памятью представляет собой семерку. Подробное описание компонентов магазинного автомата представлено в таблице 4.3.

Таблица 4.3 – Описание компонентов магазинного автомата

|  |  |  |
| --- | --- | --- |
| Компонента | Определение | Описание |
|  | Множество состояний автомата | Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата |
|  | Алфавит входных символов | Алфавит представляет из себя множества терминальных и нетерминальных символов, описание которых содержится в таблица 3.1 и 4.1. |
|  | Алфавит специальных магазинных символов | Алфавит магазинных символов содержит стартовый символ и маркер дна стека (представляет из себя символ $) |
|  | Функция переходов автомата | Функция представляет из себя множество правил грамматики, описанных в таблице 4.1. |
|  | Начальное состояние автомата | Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики |

Таблица 4.3 продолжение

|  |  |  |
| --- | --- | --- |
|  | Начальное состояние магазина автомата | Символ маркера дна стека $ |
|  | Множество конечных состояний | Конечные состояние заставляют автомат прекратить свою работу. Конечным состоянием является пустой магазин автомата и совпадение позиции на входной ленте автомата с размером ленты |

## 4.4 Основные структуры данных

Основные структуры данных синтаксического анализатора представляются в виде структуры магазинного конечного автомата, выполняющего разбор исходной ленты, и структуры грамматики Грейбах, описывающей синтаксические правила языка SNE-2023. Данные структуры в приложении В.

## 4.5 Описание алгоритма синтаксического разбора

Принцип работы автомата следующий:

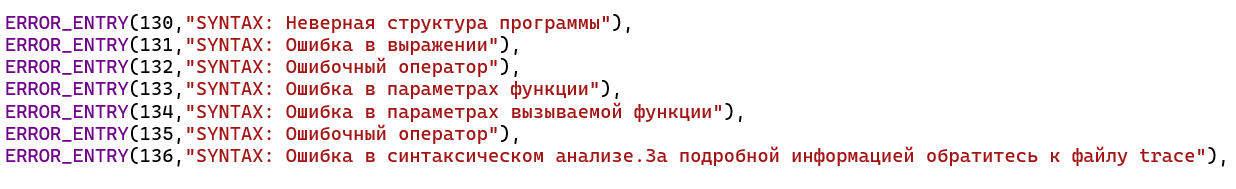
1. В магазин записывается стартовый символ;
2. На основе полученных ранее таблиц формируется входная лента;
3. Запускается автомат;
4. Выбирается цепочка, соответствующая нетерминальному символу, записывается в магазин в обратном порядке;
5. Если терминалы в стеке и в ленте совпадают, то данный терминал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

## 4.6 Параметры синтаксического анализатора и режимы его работы

Входной информацией для синтаксического анализатора является таблица лексем и идентификаторов. Кроме того, используется описание грамматики в форме Грейбах. Результаты работы синтаксического разбора, а именно дерево разбора и протокол работы автомата с магазинной памятью выводятся в журнал работы программы.

## 4.7 Структура и перечень сообщений синтаксического анализатора

Перечень ошибок синтаксического анализатора представлен на рисунке 4.4.

Рисунок 4.4 Сообщения синтаксического анализатора

## 4.8. Принцип обработки ошибок

Синтаксический анализатор выполняет разбор исходной последовательности лексем до тех пор, пока не дойдёт до конца цепочки лексем или не найдёт ошибку. Тогда анализ останавливается и выводится сообщение об ошибке (если она найдена).

## 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 анализатор принимает на свой вход результаты работ лексического и синтаксического анализаторов, то есть таблицы лексем, идентификаторов и результат работы синтаксического анализатора, то есть дерево разбора, и последовательно ищет необходимые ошибки. Общая структура обособленно работающего (не параллельно с лексическим анализом) семантического анализатора представлена на рисунке 5.1.

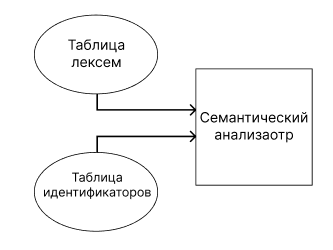


Рисунок 5.1. Структура семантического анализатора

## 5.2 Функции семантического анализатора

В языке программирования SNE-2023 выполняются следующие семантические проверки:

1. Наличие функции **start** – точки входа в программу;
2. Единственность точки входа;
3. Неправильные вычисления (например деление на 0)
4. Подсчет количество параметров специальных методов
5. Проверка типа параметров специального метода
6. Подсчет количества операндов в выражении (не> 8)
7. Проверка типа возвращаемого значение
8. Проверка что метод возвращает значение (все методы включая start должны возвращать значение)
9. Проверка максимальное количество передаваемых параметров при определении функции
10. Проверка максимальное количество передаваемых параметров при вызове функции
11. Проверка типа параметров
12. Проверка количества параметров

## 5.3 Структура и перечень сообщений семантического анализатора

Сообщения, формируемые семантическим анализатором, представлены на рисунке 5.2.

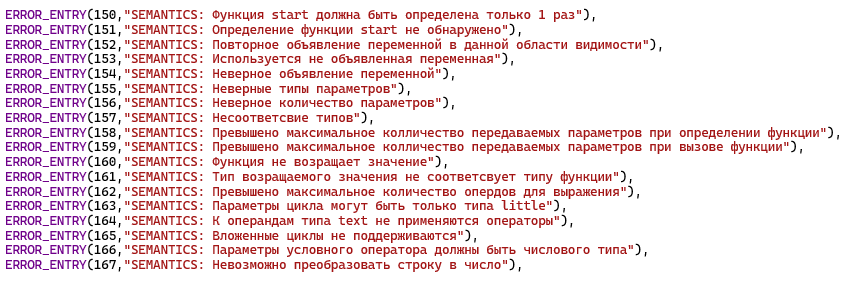


Рисунок 5.2 Перечень сообщений семантического анализатора

## 5.4 Принцип обработки ошибок

Ошибки, возникающие в процессе трансляции программы, фиксируются в протокол, заданный входным параметрами. В случае возникновения ошибок происходит их протоколирование с номером ошибки и диагностическим сообщением.

## 5.5 Контрольный пример

Соответствие примеров некоторых ошибок в исходном коде и диагностических сообщений об ошибках приведено в таблице 5.3.

Таблица 5.3. Примеры диагностики ошибок

|  |  |
| --- | --- |
| Исходный код | Текст сообщения |
| start  {  return 0;  };  start  {  return 0;  }; | Ошибка №150: Семантическая ошибка: Функция start должна быть определена только 1 раз |
| main  {  return 0;  } | Ошибка №151: Семантическая ошибка: Определение функции start не обнаруженo |
| new text n;  N = 123; | Ошибка №157: Несоответсвие типов |

В данной таблице представлены не все варианты ошибок, которые обнаруживает семантический анализатор.

# 6. Вычисление выражений

## 6.1 Выражения, допускаемые языком

В языке SNE-2023 допускаются вычисления выражений целочисленного типа данных с поддержкой вызова функций внутри выражений. Приоритет операций представлен на таблице 6.1.

Таблица 6.1. Приоритеты операций

|  |  |
| --- | --- |
| Операция | Значение приоритета |
| ( | +2 |
| ) | -2 |
| \* | +1 |
| / | +1 |
| % | +1 |
| + | 0 |
| - | 0 |

## 6.2 Польская запись и принцип её построения

Все выражения языка SNE-2023 преобразовываются к обратной польской записи.

Польская запись – это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок. Существует два типа польской записи: 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 польской записи:

* исходная строка: выражение;
* результирующая строка: польская запись;
* стек: пустой;
* исходная строка просматривается слева направо;
* операнды переносятся в результирующую строку;
* операция записывается в стек, если стек пуст;
* операция выталкивает все операции с большим или равным приоритетом в результирующую строку;
* открывающая скобка увеличивает приоритетность на 2;
* закрывающая скобка уменьшает приоритетность на 2;

## 6.3 Программная реализация обработки выражений

Программная реализация алгоритма преобразования выражений к польской записи представлена в приложении Г.

## 6.4 Контрольный пример

Пример преобразования выражений из контрольных примеров к обратной польской записи представлен в таблице 6.2. Преобразование выражений в формат польской записи необходимо для построения более простых алгоритмов их вычисления и преобразования к ассемблерному коду. В приложении Г приведены изменённые таблицы лексем и идентификаторов, отображающие результаты преобразования выражений в польский формат.

Таблица 6.2. Преобразование выражений к ПОЛИЗ

|  |  |
| --- | --- |
| Выражение | Обратная польская запись для выражения |
| i = 2+2\*2 | i = 222\*+ |
| i = (2+2)\*2 | i = 22+2\* |
| i = (2+2)-2\*2 | i = 22+22\*- |

# 7. Генерация кода

## 7.1 Структура генератора кода

Генерация кода является заключительным этапом трансляции. Генератор принимает на вход таблицы лексем и идентификаторов, полученные в результате лексического анализа. В языке SNE-2023 в соответствии с таблицей лексем строится выходной файл на языке ассемблера, который будет являться результатом работы транслятора. В случае возникновения ошибок генерация кода не будет осуществляться. Структура генератора кода SNE-2023 представлена на рисунке 7.1.

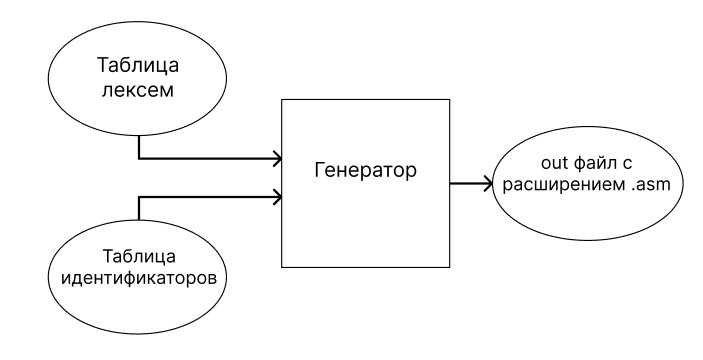


Рисунок 7.1 – Структура генератора кода

## 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 SNE-2023 и на языке ассемблера приведены в таблице 7.2.

Таблица 7.2 – Соответствия типов идентификаторов языка SNE-2023 и языка ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке SNE-2023 | Тип идентификатора на языке ассемблера | Пояснение |
| little | sdword | Хранит целочисленный тип данных. |
| text | byte | Хранит указатель на начало строки. |

## 7.3 Статическая библиотека

В языке SNE-2023 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++. Объявление функций статической библиотеки генерируется автоматически в коде ассемблера.

Таблица 7.3 – Функции статической библиотеки

|  |  |
| --- | --- |
| Функция | Назначение |
| int outtxt(char\* value) | Вывод на консоль строки text |
| int outlit(int value) | Вывод на консоль целочисленной переменной little |
| int textlenght(char\* str1) | Вычисление длины строки |
| char\* copytxt(char\* str1, char\* str2) | Копирование строки str1 в str2 |
| int texttolit(char\* str1) | Преобразование строки в число |

## 7.4 Особенности алгоритма генерации кода

В языке SNE-2023 генерация кода строится на основе таблиц лексем и идентификаторов. Общая схема работы генератора кода представлена на рисунке 7.4.

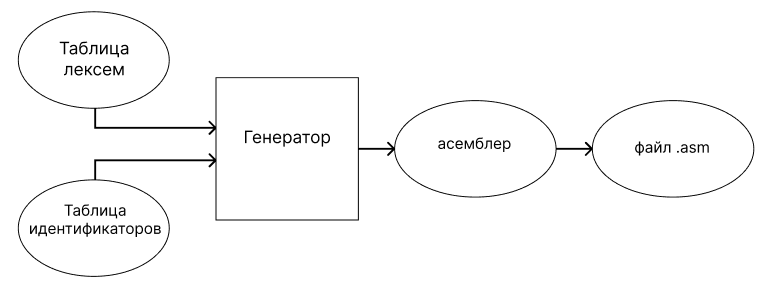


Рисунок 7.4 Структура генератора кода

## 7.5 Параметры, управляющие генерацией кода

На вход генератору кода поступают таблицы лексем и идентификаторов исходного код программы на языке SNE-2023. Результаты работы генератора кода выводятся в файл с расширением .asm.

## 7.6 Контрольный пример

Результат генерации ассемблерного кода на основе контрольного примера из приложения А приведен в приложении Д. Результат работы контрольного примера приведён на рисунке 7.5.

|  |
| --- |
|  |

Рисунок 7.5 Результат работы программы на языке SNE-2023

Здесь приведен вывод на консоль результат работы кода на языке SNE-2023 который прошел процесс компиляции.

# 8. Тестирование транслятора

## 8.1 Общие приложения

В языке SNE-2023 не разрешается использовать запрещённые входным алфавитом символы. Результат использования запрещённого символа показан в таблице 8.1.

На этапе лексического анализа в языке SNE-2023 могут возникнуть ошибки, описанные в пункте 3.10. Результаты тестирования лексического анализатора показаны в таблице 8.1.

На этапе синтаксического анализа в языке SNE-2023 могут возникнуть ошибки, описанные в пункте 4.7. Результаты тестирования синтаксического анализатора показаны в таблице 8.1.

Семантический анализ в языке SNE-2023 содержит множество проверок по семантическим правилам, описанным в пункте 1.16. Итоги тестирования семантического анализатора на корректное обнаружение семантических ошибок приведены в таблице 8.1.

## 8.2 Результаты тестирования

Таблица 8.1 - Тестирование

|  |  |
| --- | --- |
| Тестирование на допустимость символов | |
| Исходный код | Диагностическое сообщение |
| start  {  output(Привет);  return 0;  }; | Ошибка 102: PARAMETERS: Недопустимый символ в исходном файле (-in), строка 3 |
| Тестирование лексического анализатора | |
| start  {  new text h;  h = ‘hello;  return 0;  }; | Ошибка 112: LEXICAL: Не найден конец строкового литерала, строка 4 |
| start  {  new text Hi;  return 0;  }; | Ошибка 117: LEXICAL: Имя идентификатора задано неверно, строка 3 |

Таблица 8.1 - продолжение

|  |  |
| --- | --- |
| start  {  new little i;  i = !18;  return 0;  }; | Ошибка 120: LEXICAL: Неправильно объявление числового литерала в восьмерично представлении, строка 4 |
| Тестирование синтаксического анализатора | |
| start  {  new little n;  n = 2++2;  return 0;  }; | 132: строка 4, SYNTAX: Ошибочный оператор |
| little function h(text ) | 133: строка 1, SYNTAX: Ошибка в параметрах функции |
| little function h(little g)  {  return 0;  };  start  {  g = h(text);  return 0;  }; | 134: строка 7, SYNTAX: Ошибка в параметрах вызываемой функции |
| Тестирование семантического анализатора | |
| text function hi()  {  return 0;  }; | SEMANTICS: Функция start должна быть определена только 1 раз |
| start{return 0;};  start{return 0;}; | SEMANTICS: Определение функции start не обнаруженo |
| start  {  new little g;  new little g;  return 0;  }; | SEMANTICS: Повторное объявление переменной в данной области видимости |

Таблица 8.1 - продолжение

|  |  |
| --- | --- |
| start  {  new little g;  g = ‘hi’;  return 0;  };  Исходный код | Ошибка 157: SEMANTICS: Несоответствие типов, строка 4  Диагностическое сообщение |
| start  {  check(2 = '2')?  {  output('yes');  }  return 0;  }; | Ошибка 166: SEMANTICS: Параметры условного оператора должны быть числового типа, строка 3 |

# Заключение

В ходе выполнения курсовой работы был разработан транслятор и генератор кода для языка программирования SNE-2023 со всеми необходимыми компонентами. Таким образом, были выполнены основные задачи данной курсовой работы:

1. Сформулирована спецификация языка SNE-2023;
2. Разработаны конечные автоматы и важные алгоритмы на их основе для эффективной работы лексического анализатора;
3. Осуществлена программная реализация лексического анализатора, распознающего допустимые цепочки спроектированного языка;
4. Разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
5. Осуществлена программная реализация синтаксического анализатора;
6. Разработан семантический анализатор, осуществляющий проверку используемых инструкций на соответствие логическим правилам;
7. Разработан транслятор кода на язык ассемблера;
8. Проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка SNE-2023 включает:

1. 2 типа данных;
2. Поддержка операторов вывода строки;
3. Возможность вызова функций стандартной библиотеки;
4. Наличие 5 арифметических операторов для вычисления выражений;
5. Поддержка функций и условия;
6. Структурированная и классифицированная система для обработки ошибок пользователя.

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

# Графический материал

Дерево разбора

|  |
| --- |
|  |

# Список использованных источников

1. Курс лекций по ЯП / Наркевич А.С.

2. Креншоу, Д. Давайте создадим компилятор! / Д. Креншоу. – 3-е изд. – Моска: 1995 – 73 с.

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

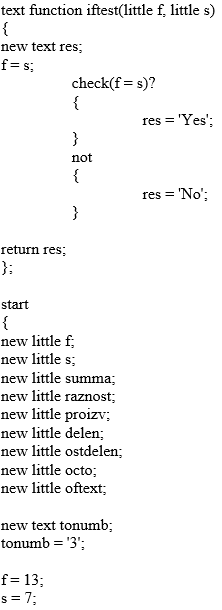
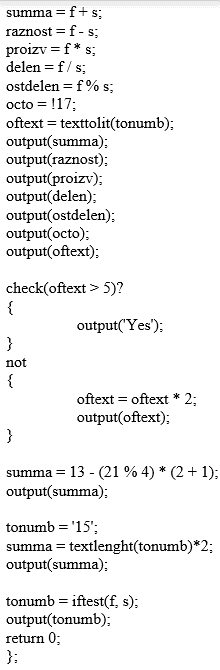
4. Прата, С. Язык программирования С++. Лекции и упражнения / С. Прата. – М., 2006 — 1104 c.

5. Пратт, Т. Языки программирования: разработка и реализация / Т. Пратт – 4-е изд., 2002 – 690 с.

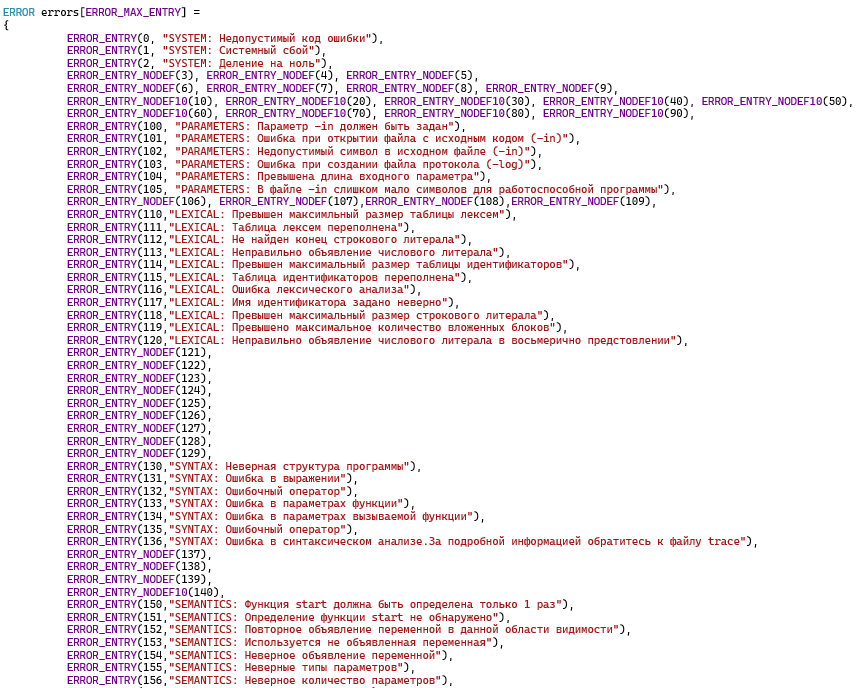
6. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. – 3-е изд. – Москва : Вильямс, 2003. - 429 с.

# Приложение А

Листинг 1 Исходный код программы на языка SNE-2023

Листинг 2 Таблица ошибок



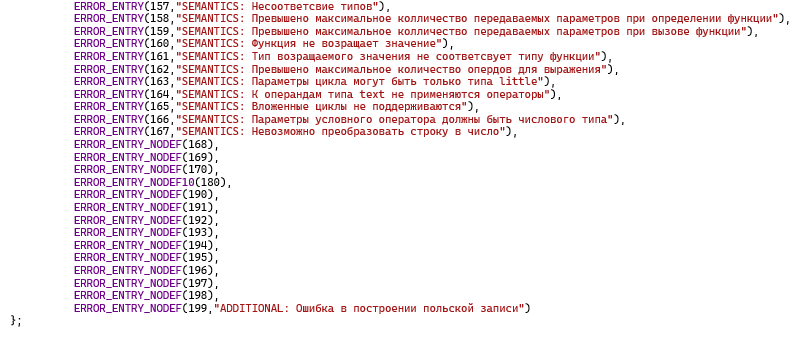


Рисунок 1 Таблица ошибок языка SNE-2023

# Приложение Б

Листинг 1 Таблица идентификаторов контрольного примера

|  |
| --- |
| =================ID TABLE==============  Name Type Data type First in LT Value Visibility  0 buffer variable little -1 0 0  1 iftest function text 2 NV 0  2 f parameter little 5 0 0 1  3 s parameter little 8 0 0 1  4 res variable text 13 NV 0 1  5 T0 literal text 29 'Yes' 0 1 2  6 T1 literal text 36 'No' 0 1 3  7 start function little 44 0 0  8 f variable little 48 0 0 4  9 s variable little 52 0 0 4  10 summa variable little 56 0 0 4  11 raznost variable little 60 0 0 4  12 proizv variable little 64 0 0 4  13 delen variable little 68 0 0 4  14 ostdelen variable little 72 0 0 4  15 octo variable little 76 0 0 4  16 oftext variable little 80 0 0 4  17 tonumb variable text 84 NV 0 4  18 T2 literal text 88 '3' 0 4  19 L0 literal little 92 13 0 4  20 L1 literal little 96 7 0 4  21 L2 literal little 130 15 0 4  22 L3 literal little 178 5 0 4  23 T3 literal text 184 'Yes' 0 4 5  24 L4 literal little 193 2 0 4 6  25 L5 literal little 204 13 0 4  26 L6 literal little 205 21 0 4  27 L7 literal little 206 4 0 4  28 L8 literal little 208 2 0 4  29 L9 literal little 209 1 0 4  30 T4 literal text 225 '15' 0 4  31 L10 literal little 232 2 0 4  32 L11 literal little 256 0 0 4 |

Листинг 2 Таблица лексем после контрольного примера

|  |  |
| --- | --- |
| ================LEX TABLE============== | |
| Num Line Lexema ID in IT value  0 1 t t  1 1 f  2 1 i 1  3 1 (  4 1 t l  5 1 i 2  6 1 ,  7 1 t l  8 1 i 3  9 1 )  10 2 {  11 3 n  12 3 t t  13 3 i 4  14 3 ;  15 4 i 2  16 4 =  17 4 i 3  18 4 ;  19 5 c  20 5 (  21 5 i 2  22 5 =  23 5 i 3  24 5 )  25 5 ? ?  26 6 {  27 7 i 4  28 7 =  29 7 l 5  30 7 ;  31 8 }  32 10 j  33 10 {  34 11 i 4  35 11 =  36 11 l 6  37 11 ;  38 12 }  39 14 r  40 14 i 4  41 14 ;  42 15 }  43 15 ;  44 17 s 7  45 18 {  46 19 n  47 19 t l  48 19 i 8  49 19 ;  50 20 n  51 20 t l  52 20 i 9  53 20 ;  54 21 n  55 21 t l  56 21 i 10  57 21 ;  58 22 n  59 22 t l  60 22 i 11  61 22 ;  62 23 n  63 23 t l  64 23 i 12  65 23 ;  66 24 n  67 24 t l  68 24 i 13  69 24 ;  70 25 n  71 25 t l  72 25 i 14  73 25 ;  74 26 n  75 26 t l  76 26 i 15  77 26 ;  78 27 n  79 27 t l  80 27 i 16  81 27 ;  82 29 n  83 29 t t  84 29 i 17  85 29 ;  86 30 i 17  87 30 =  88 30 l 18  89 30 ;  90 32 i 8  91 32 =  92 32 l 19  93 32 ;  94 33 i 9  95 33 =  96 33 l 20  97 33 ;  98 35 i 10  99 35 =  100 35 i 8  101 35 i 9  102 35 v +  103 35 ;  104 36 i 11  105 36 =  106 36 i 8  107 36 i 9  108 36 v -  109 36 ;  110 37 i 12  111 37 =  112 37 i 8  113 37 i 9  114 37 v \*  115 37 ;  116 38 i 13  117 38 =  118 38 i 8  119 38 i 9  120 38 v / | 121 38 ;  122 39 i 14  123 39 =  124 39 i 8  125 39 i 9  126 39 v %  127 39 ;  128 40 i 15  129 40 =  130 40 l 21  131 40 ;  132 41 i 16  133 41 =  134 41 i 17  135 41 @ -3 z  136 41 1 -1 z  137 41 # -1  138 41 # -1  139 42 o  140 42 (  141 42 i 10  142 42 )  143 42 ;  144 43 o  145 43 (  146 43 i 11  147 43 )  148 43 ;  149 44 o  150 44 (  151 44 i 12  152 44 )  153 44 ;  154 45 o  155 45 (  156 45 i 13  157 45 )  158 45 ;  159 46 o  160 46 (  161 46 i 14  162 46 )  163 46 ;  164 47 o  165 47 (  166 47 i 15  167 47 )  168 47 ;  169 48 o  170 48 (  171 48 i 16  172 48 )  173 48 ;  174 50 c  175 50 (  176 50 i 16  177 50 y >  178 50 l 22  179 50 )  180 50 ? ?  181 51 {  182 52 o  183 52 (  184 52 l 23  185 52 )  186 52 ;  187 53 }  188 55 j  189 55 {  190 56 i 16  191 56 =  192 56 i 16  193 56 l 24  194 56 v \*  195 56 ;  196 57 o  197 57 (  198 57 i 16  199 57 )  200 57 ;  201 58 }  202 60 i 10  203 60 =  204 60 l 25  205 60 l 26  206 60 l 27  207 60 v %  208 60 l 28  209 60 l 29  210 60 v +  211 60 v \*  212 60 v -  213 60 ;  214 1 # -1  215 1 # -1  216 1 # -1  217 1 # -1  218 61 o  219 61 (  220 61 i 10  221 61 )  222 61 ;  223 63 i 17  224 63 =  225 63 l 30  226 63 ;  227 64 i 10  228 64 =  229 64 i 17  230 64 @ -2 z  231 64 1 -1 z  232 64 l 31  233 64 v \*  234 64 # -1  235 64 # -1  236 65 o  237 65 (  238 65 i 10  239 65 )  240 65 ;  241 67 i 17  242 67 =  243 67 i 8  244 67 i 9  245 67 @ 1  246 67 2 -1  247 67 ;  248 67 # -1  249 67 # -1  250 68 o  251 68 (  252 68 i 17  253 68 )  254 68 ;  255 69 r  256 69 l 32  257 69 ;  258 70 }  259 70 ; |

Листинг 2 Структура магазинного автомата

|  |
| --- |
| struct MfstState  {  short lenta\_position;  short nrule;  short nrulechain;  MFSTSTACK st;  MfstState();  MfstState(short pposition, MFSTSTACK pst, short pnrulechain);  MfstState(short pposition, MFSTSTACK pst, short pnrule, short pnrulechain);  };  struct Mfst  {  enum RC\_STEP {  NS\_OK,  NS\_NORULE,  NS\_NORULECHAIN,  NS\_ERROR,  TS\_OK,  TS\_NOK,  LENTA\_END,  SURPRISE  };  struct MfstDiagnosis  {  short lenta\_position;  RC\_STEP rc\_step;  short nrule;  short nrule\_chain;  MfstDiagnosis();  MfstDiagnosis(short plenta\_position, RC\_STEP prt\_step, short pnrule, short pnrule\_chain);  } diagnosis[MFST\_DIAGN\_NUMBER];  GRBALPHABET\* lenta;  short lenta\_position;  short nrule;  short nrulechain;  short lenta\_size;  GRB::Greibach grebach;  LexA::Tables lex;  MFSTSTACK st;  std::stack<MfstState> storestate;  Mfst();  Mfst(LexA::Tables plex, GRB::Greibach pgrebach);  char\* getCSt(char\* buf);  char\* getCLenta(char\* buf, short pos, short n = 25);  char\* getDiagnosis(short n, char\* buf);  bool savestate(std::ofstream& log);  bool reststate(std::ofstream& log);  bool push\_chain(GRB::Rule::Chain chain);  RC\_STEP step(std::ofstream& log);  bool start(std::ofstream& log);  bool savediagnosis(RC\_STEP pprc\_step);  void printrules(Log::LOG log); |

Листинг 3 Структура грамматики Грейбах

|  |
| --- |
| struct Rule  {  GRBALPHABET nn;  int iderror;  short size;  struct Chain  {  short size;  GRBALPHABET\* nt;  Chain()  {  size = 0;  nt = 0;  };  Chain(short psize, GRBALPHABET s, ...);  char\* getCChain(char\* b);  static GRBALPHABET T(char t)  {  return GRBALPHABET(t);  };  static GRBALPHABET N(char n)  {  return -GRBALPHABET(n);  };  static bool isT(GRBALPHABET s)  {  return s > 0;  };  static bool isN(GRBALPHABET s)  {  return !isT(s);  };  static char alphabet\_to\_char(GRBALPHABET s)  {  return isT(s) ? char(s) : char(-s);  };  }\*chains;  Rule()  {  nn = 0x00;  size = 0;  };  Rule(GRBALPHABET pnn, int iderror, short psize, Chain c, ...);  char\* getCRule(char\* b, short nchain);  short getNextChain(GRBALPHABET t, Rule::Chain& pchain, short j);  }; |

Листинг 4 Разбор исходного кода синтаксическим анализатором

|  |  |
| --- | --- |
| Шаг : Правило Входная лента Стек  0 : S->tfi(){N}; tfi(ti,ti){nti;i=i;c(i=i) S$  0 : SAVESTATE: 1  0 : tfi(ti,ti){nti;i=i;c(i=i) tfi(){N};$  1 : fi(ti,ti){nti;i=i;c(i=i)? fi(){N};$  2 : i(ti,ti){nti;i=i;c(i=i)?{ i(){N};$  3 : (ti,ti){nti;i=i;c(i=i)?{i (){N};$  4 : ti,ti){nti;i=i;c(i=i)?{i= ){N};$  5 : TS\_NOK/NS\_NORULECHAIN  5 : RESSTATE  5 : tfi(ti,ti){nti;i=i;c(i=i) S$  6 : S->tfi(F){N}; tfi(ti,ti){nti;i=i;c(i=i) S$  6 : SAVESTATE: 1  6 : tfi(ti,ti){nti;i=i;c(i=i) tfi(F){N};$  7 : fi(ti,ti){nti;i=i;c(i=i)? fi(F){N};$  8 : i(ti,ti){nti;i=i;c(i=i)?{ i(F){N};$  9 : (ti,ti){nti;i=i;c(i=i)?{i (F){N};$  10 : ti,ti){nti;i=i;c(i=i)?{i= F){N};$  11 : F->ti ti,ti){nti;i=i;c(i=i)?{i= F){N};$  11 : SAVESTATE: 2  11 : ti,ti){nti;i=i;c(i=i)?{i= ti){N};$  12 : i,ti){nti;i=i;c(i=i)?{i=l i){N};$  13 : ,ti){nti;i=i;c(i=i)?{i=l; ){N};$  14 : TS\_NOK/NS\_NORULECHAIN  14 : RESSTATE  14 : ti,ti){nti;i=i;c(i=i)?{i= F){N};$  15 : F->ti,F ti,ti){nti;i=i;c(i=i)?{i= F){N};$  15 : SAVESTATE: 2  15 : ti,ti){nti;i=i;c(i=i)?{i= ti,F){N};$  16 : i,ti){nti;i=i;c(i=i)?{i=l i,F){N};$  17 : ,ti){nti;i=i;c(i=i)?{i=l; ,F){N};$  18 : ti){nti;i=i;c(i=i)?{i=l;} F){N};$  19 : F->ti ti){nti;i=i;c(i=i)?{i=l;} F){N};$  19 : SAVESTATE: 3  19 : ti){nti;i=i;c(i=i)?{i=l;} ti){N};$  20 : i){nti;i=i;c(i=i)?{i=l;}j i){N};$  21 : ){nti;i=i;c(i=i)?{i=l;}j{ ){N};$  22 : {nti;i=i;c(i=i)?{i=l;}j{i {N};$  23 : nti;i=i;c(i=i)?{i=l;}j{i= N};$  24 : N->nti; nti;i=i;c(i=i)?{i=l;}j{i= N};$  24 : SAVESTATE: 4  24 : nti;i=i;c(i=i)?{i=l;}j{i= nti;};$  25 : ti;i=i;c(i=i)?{i=l;}j{i=l ti;};$  26 : i;i=i;c(i=i)?{i=l;}j{i=l; i;};$  27 : ;i=i;c(i=i)?{i=l;}j{i=l;} ;};$  28 : i=i;c(i=i)?{i=l;}j{i=l;}r };$  29 : TS\_NOK/NS\_NORULECHAIN  29 : RESSTATE  29 : nti;i=i;c(i=i)?{i=l;}j{i= N};$  30 : N->nti;N nti;i=i;c(i=i)?{i=l;}j{i= N};$  30 : SAVESTATE: 4  30 : nti;i=i;c(i=i)?{i=l;}j{i= nti;N};$  31 : ti;i=i;c(i=i)?{i=l;}j{i=l ti;N};$  32 : i;i=i;c(i=i)?{i=l;}j{i=l; i;N};$  33 : ;i=i;c(i=i)?{i=l;}j{i=l;} ;N};$  34 : i=i;c(i=i)?{i=l;}j{i=l;}r N};$  35 : N->i=E; i=i;c(i=i)?{i=l;}j{i=l;}r N};$  35 : SAVESTATE: 5  35 : i=i;c(i=i)?{i=l;}j{i=l;}r i=E;};$  36 : =i;c(i=i)?{i=l;}j{i=l;}ri =E;};$  37 : i;c(i=i)?{i=l;}j{i=l;}ri; E;};$  38 : E->i i;c(i=i)?{i=l;}j{i=l;}ri; E;};$  38 : SAVESTATE: 6  38 : i;c(i=i)?{i=l;}j{i=l;}ri; i;};$  39 : ;c(i=i)?{i=l;}j{i=l;}ri;} ;};$  40 : c(i=i)?{i=l;}j{i=l;}ri;}; };$  41 : TS\_NOK/NS\_NORULECHAIN  41 : RESSTATE  41 : i;c(i=i)?{i=l;}j{i=l;}ri; E;};$  42 : E->i(W) i;c(i=i)?{i=l;}j{i=l;}ri; E;};$  42 : SAVESTATE: 6  42 : i;c(i=i)?{i=l;}j{i=l;}ri; i(W);};$  43 : ;c(i=i)?{i=l;}j{i=l;}ri;} (W);};$  44 : TS\_NOK/NS\_NORULECHAIN  44 : RESSTATE  44 : i;c(i=i)?{i=l;}j{i=l;}ri; E;};$  45 : E->i() i;c(i=i)?{i=l;}j{i=l;}ri; E;};$  45 : SAVESTATE: 6  45 : i;c(i=i)?{i=l;}j{i=l;}ri; i();};$  46 : ;c(i=i)?{i=l;}j{i=l;}ri;} ();};$  47 : TS\_NOK/NS\_NORULECHAIN  47 : RESSTATE  47 : i;c(i=i)?{i=l;}j{i=l;}ri; E;};$  48 : E->iM i;c(i=i)?{i=l;}j{i=l;}ri; E;};$  48 : SAVESTATE: 6  48 : i;c(i=i)?{i=l;}j{i=l;}ri; iM;};$  49 : ;c(i=i)?{i=l;}j{i=l;}ri;} M;};$  50 : TNS\_NORULECHAIN/NS\_NORULE  50 : RESSTATE  50 : i;c(i=i)?{i=l;}j{i=l;}ri; E;};$ | 2017: ,i);o(i);rl;}; ,W)M;};$  2018: i);o(i);rl;}; W)M;};$  2019: W->i i);o(i);rl;}; W)M;};$  2019: SAVESTATE: 111  2019: i);o(i);rl;}; i)M;};$  2020: );o(i);rl;}; )M;};$  2021: ;o(i);rl;}; M;};$  2022: TNS\_NORULECHAIN/NS\_NORULE  2022: RESSTATE  2022: i);o(i);rl;}; W)M;};$  2023: W->i,W i);o(i);rl;}; W)M;};$  2023: SAVESTATE: 111  2023: i);o(i);rl;}; i,W)M;};$  2024: );o(i);rl;}; ,W)M;};$  2025: TS\_NOK/NS\_NORULECHAIN  2025: RESSTATE  2025: i);o(i);rl;}; W)M;};$  2026: TNS\_NORULECHAIN/NS\_NORULE  2026: RESSTATE  2026: i,i);o(i);rl;}; W)M;};$  2027: TNS\_NORULECHAIN/NS\_NORULE  2027: RESSTATE  2027: i(i,i);o(i);rl;}; E;};$  2028: E->i()M i(i,i);o(i);rl;}; E;};$  2028: SAVESTATE: 109  2028: i(i,i);o(i);rl;}; i()M;};$  2029: (i,i);o(i);rl;}; ()M;};$  2030: i,i);o(i);rl;}; )M;};$  2031: TS\_NOK/NS\_NORULECHAIN  2031: RESSTATE  2031: i(i,i);o(i);rl;}; E;};$  2032: TNS\_NORULECHAIN/NS\_NORULE  2032: RESSTATE  2032: i=i(i,i);o(i);rl;}; N};$  2033: N->i=E;N i=i(i,i);o(i);rl;}; N};$  2033: SAVESTATE: 108  2033: i=i(i,i);o(i);rl;}; i=E;N};$  2034: =i(i,i);o(i);rl;}; =E;N};$  2035: i(i,i);o(i);rl;}; E;N};$  2036: E->i i(i,i);o(i);rl;}; E;N};$  2036: SAVESTATE: 109  2036: i(i,i);o(i);rl;}; i;N};$  2037: (i,i);o(i);rl;}; ;N};$  2038: TS\_NOK/NS\_NORULECHAIN  2038: RESSTATE  2038: i(i,i);o(i);rl;}; E;N};$  2039: E->i(W) i(i,i);o(i);rl;}; E;N};$  2039: SAVESTATE: 109  2039: i(i,i);o(i);rl;}; i(W);N};$  2040: (i,i);o(i);rl;}; (W);N};$  2041: i,i);o(i);rl;}; W);N};$  2042: W->i i,i);o(i);rl;}; W);N};$  2042: SAVESTATE: 110  2042: i,i);o(i);rl;}; i);N};$  2043: ,i);o(i);rl;}; );N};$  2044: TS\_NOK/NS\_NORULECHAIN  2044: RESSTATE  2044: i,i);o(i);rl;}; W);N};$  2045: W->i,W i,i);o(i);rl;}; W);N};$  2045: SAVESTATE: 110  2045: i,i);o(i);rl;}; i,W);N};$  2046: ,i);o(i);rl;}; ,W);N};$  2047: i);o(i);rl;}; W);N};$  2048: W->i i);o(i);rl;}; W);N};$  2048: SAVESTATE: 111  2048: i);o(i);rl;}; i);N};$  2049: );o(i);rl;}; );N};$  2050: ;o(i);rl;}; ;N};$  2051: o(i);rl;}; N};$  2052: N->o(U);N o(i);rl;}; N};$  2052: SAVESTATE: 112  2052: o(i);rl;}; o(U);N};$  2053: (i);rl;}; (U);N};$  2054: i);rl;}; U);N};$  2055: U->i i);rl;}; U);N};$  2055: SAVESTATE: 113  2055: i);rl;}; i);N};$  2056: );rl;}; );N};$  2057: ;rl;}; ;N};$  2058: rl;}; N};$  2059: N->rU; rl;}; N};$  2059: SAVESTATE: 114  2059: rl;}; rU;};$  2060: l;}; U;};$  2061: U->l l;}; U;};$  2061: SAVESTATE: 115  2061: l;}; l;};$  2062: ;}; ;};$  2063: }; };$  2064: ; ;$  2065: $  2066: LENTA\_END  2067: ------>LENTA\_END |

# Приложение Г

Листинг 1 Программная реализация механизма преобразования в ПОЛИЗ

|  |
| --- |
| bool polishNotation(int lextable\_pos, LT::LexTable& lextable, IT::IdTable& idtable)  {  short Line = 0;  short counter = 0;  short buffershort;  short countOfAllInputLex = 0;  short countOfOperators;  short countOfParameters = 0;  bool isAFunction = false;  int startLex = lextable\_pos;  do  {  countOfAllInputLex++;  if (lextable.table[lextable\_pos].lexema != LEX\_LEFTTHESIS &&  lextable.table[lextable\_pos].lexema != LEX\_RIGHTTHESIS)  counter++;  if (lextable.table[lextable\_pos].lexema == LEX\_COMMA &&  lextable.table[lextable\_pos - 2].lexema != LEX\_LEFTTHESIS)  counter++;  lextable\_pos++;  } while (lextable.table[lextable\_pos].lexema != LEX\_SEMICOLON);  LT::Entry bufferEntry;  LT::Entry\* AlmostAllEntries = new LT::Entry[32];  short counter1 = 0;  LT::Entry\* inputOperEntries = new LT::Entry[16];  short counter2 = 0;  short\* litPositions = new short[32];  short counter3 = 0;  short\* operatorPriority = new short[16];  short currentPriority = 0;  short maxPriority = 1;  countOfOperators = 0;  lextable\_pos = startLex;  buffershort = 0;  for (int i = 0; (i < countOfAllInputLex + 1) || lextable.table[startLex + i - 1].lexema == LEX\_SEMICOLON; i++)  {  if (lextable.table[startLex + i].lexema == LEX\_ID)  if (lextable.table[startLex + i].value == LEX\_LIBFUNCTION||  idtable.table[lextable.table[startLex + i].idxTI].idtype == IT::F)  isAFunction = true;  switch (lextable.table[lextable\_pos + i].lexema)  {  case LEX\_LEFTTHESIS:  currentPriority = currentPriority + 2;  if (maxPriority <= currentPriority)  maxPriority = currentPriority + 1;  continue;  case LEX\_RIGHTTHESIS:  currentPriority = currentPriority - 2;  continue;  case LEX\_OPERATOR:  inputOperEntries[counter2] = lextable.table[lextable\_pos + i];  operatorPriority[counter2] = currentPriority;  inputOperEntries[counter2].index = buffershort;  if (lextable.table[lextable\_pos + i].value == '\*' ||  lextable.table[lextable\_pos + i].value == '/' ||  lextable.table[lextable\_pos + i].value == '%')  operatorPriority[counter2]++;  counter2++;  AlmostAllEntries[counter1] = lextable.table[lextable\_pos + i];  AlmostAllEntries[counter1].index = buffershort;  counter1++;  buffershort++;  countOfOperators++;  continue;  default:  if (isAFunction)  {  countOfParameters = 0;  bufferEntry = lextable.table[startLex + i];  bufferEntry.lexema = '@';  i++;  Line = lextable.table[lextable\_pos + i].sn;  while (lextable.table[lextable\_pos + i].lexema != LEX\_RIGHTTHESIS)  {  if (lextable.table[lextable\_pos + i].lexema == LEX\_LITERAL ||  lextable.table[lextable\_pos + i].lexema == LEX\_ID)  {  countOfParameters++;  AlmostAllEntries[counter1] = lextable.table[lextable\_pos + i];  AlmostAllEntries[counter1].index = buffershort;  AlmostAllEntries[counter1].used = true;  counter1++;  buffershort++;  }  i++;  }  AlmostAllEntries[counter1] = bufferEntry;  AlmostAllEntries[counter1].index = buffershort;  AlmostAllEntries[counter1].used = true;  counter1++;  buffershort++;  bufferEntry.idxTI = TI\_NULLIDX;  bufferEntry.used = true;  bufferEntry.index = buffershort;  bufferEntry.lexema = intToChar(countOfParameters);  AlmostAllEntries[counter1] = bufferEntry;  counter1++;  buffershort++;  isAFunction = false;  }  else  {  AlmostAllEntries[counter1] = lextable.table[lextable\_pos + i];  AlmostAllEntries[counter1].index = buffershort;  if (AlmostAllEntries[counter1].lexema == LEX\_SEMICOLON)  AlmostAllEntries[counter1].used = false;  counter1++;  buffershort++;  }  break;  }  }  if (currentPriority != 0)  {  delete[] AlmostAllEntries;  delete[] inputOperEntries;  delete[] operatorPriority;  return false;  }  int r = 0;  while (r < countOfOperators)  {  bufferEntry = findOperator(inputOperEntries, operatorPriority, countOfOperators, maxPriority);  AlmostAllEntries[bufferEntry.index - 1].used = true;  AlmostAllEntries[bufferEntry.index].used = true;  for (int i = 0; i < countOfOperators; i++)  if (inputOperEntries[i].index == bufferEntry.index)  inputOperEntries[i].used = true;  if (AlmostAllEntries[bufferEntry.index + 1].used != true)  {  AlmostAllEntries[bufferEntry.index + 1].used = true;  AlmostAllEntries[bufferEntry.index] = AlmostAllEntries[bufferEntry.index + 1];  bufferEntry.used = true;  AlmostAllEntries[bufferEntry.index + 1] = bufferEntry;  }  else  {  int p = 1;  while (AlmostAllEntries[bufferEntry.index + p].used == true)  {  AlmostAllEntries[bufferEntry.index + p - 1] = AlmostAllEntries[bufferEntry.index + p];  p++;  }  bufferEntry.used = true;  AlmostAllEntries[bufferEntry.index + p - 1] = bufferEntry;  }  r++;  }  for (int i = 0; (i < counter + 1) && AlmostAllEntries[i - 1].lexema != LEX\_SEMICOLON; i++)  {  if (AlmostAllEntries[i].lexema == LEX\_LITERAL)  litPositions[counter3++] = i;  lextable.table[startLex + i] = AlmostAllEntries[i];  buffershort = i;  }  bufferEntry.idxTI = TI\_NULLIDX;  bufferEntry.index = -1;  bufferEntry.lexema = LATTICE;  bufferEntry.used = false;  bufferEntry.value = NULL;  bufferEntry.sn = Line;  for (int k = buffershort + 1; k < countOfAllInputLex + 1; k++)  {  AlmostAllEntries[k] = bufferEntry;  lextable.table[startLex + k] = bufferEntry;  }  int k = 0;  for (int i = 0; i < counter1; i++)  if (AlmostAllEntries[i].lexema == LEX\_LITERAL)  idtable.table[AlmostAllEntries[i].idxTI].idxfirstLE = lextable\_pos + litPositions[k++];  delete[] AlmostAllEntries;  delete[] inputOperEntries;  delete[] operatorPriority;  return true;  } |

Листинг 2 Таблица идентификаторов после преобразования выражений в ПОЛИЗ

|  |
| --- |
| =================ID TABLE==============  Name Type Data type First in LT Value Visibility  0 buffer variable little -1 0 0  1 iftest function text 2 NV 0  2 f parameter little 5 0 0 1  3 s parameter little 8 0 0 1  4 res variable text 13 NV 0 1  5 T0 literal text 29 'Yes' 0 1 2  6 T1 literal text 36 'No' 0 1 3  7 start function little 44 0 0  8 f variable little 48 0 0 4  9 s variable little 52 0 0 4  10 summa variable little 56 0 0 4  11 raznost variable little 60 0 0 4  12 proizv variable little 64 0 0 4  13 delen variable little 68 0 0 4  14 ostdelen variable little 72 0 0 4  15 octo variable little 76 0 0 4  16 oftext variable little 80 0 0 4  17 tonumb variable text 84 NV 0 4  18 T2 literal text 88 '3' 0 4  19 L0 literal little 92 13 0 4  20 L1 literal little 96 7 0 4  21 L2 literal little 130 15 0 4  22 L3 literal little 178 5 0 4  23 T3 literal text 184 'Yes' 0 4 5  24 L4 literal little 193 2 0 4 6  25 L5 literal little 204 13 0 4  26 L6 literal little 205 21 0 4  27 L7 literal little 206 4 0 4  28 L8 literal little 208 2 0 4  29 L9 literal little 209 1 0 4  30 T4 literal text 225 '15' 0 4  31 L10 literal little 232 2 0 4  32 L11 literal little 256 0 0 4 |

Листинг 3 Таблица лексем после преобразования выражений в ПОЛИЗ

|  |  |
| --- | --- |
| ================LEX TABLE============== | |
| Num Line Lexema ID in IT value  0 1 t t  1 1 f  2 1 i 1  3 1 (  4 1 t l  5 1 i 2  6 1 ,  7 1 t l  8 1 i 3  9 1 )  10 2 {  11 3 n  12 3 t t  13 3 i 4  14 3 ;  15 4 i 2  16 4 =  17 4 i 3  18 4 ;  19 5 c  20 5 (  21 5 i 2  22 5 =  23 5 i 3  24 5 )  25 5 ? ?  26 6 {  27 7 i 4  28 7 =  29 7 l 5  30 7 ;  31 8 }  32 10 j  33 10 {  34 11 i 4  35 11 =  36 11 l 6  37 11 ;  38 12 }  39 14 r  40 14 i 4  41 14 ;  42 15 }  43 15 ;  44 17 s 7  45 18 {  46 19 n  47 19 t l  48 19 i 8  49 19 ;  50 20 n  51 20 t l  52 20 i 9  53 20 ;  54 21 n  55 21 t l  56 21 i 10  57 21 ;  58 22 n  59 22 t l  60 22 i 11  61 22 ;  62 23 n  63 23 t l  64 23 i 12  65 23 ;  66 24 n  67 24 t l  68 24 i 13  69 24 ;  70 25 n  71 25 t l  72 25 i 14  73 25 ;  74 26 n  75 26 t l  76 26 i 15  77 26 ;  78 27 n  79 27 t l  80 27 i 16  81 27 ;  82 29 n  83 29 t t  84 29 i 17  85 29 ;  86 30 i 17  87 30 =  88 30 l 18  89 30 ;  90 32 i 8  91 32 =  92 32 l 19  93 32 ;  94 33 i 9  95 33 =  96 33 l 20  97 33 ;  98 35 i 10  99 35 =  100 35 i 8  101 35 i 9  102 35 v +  103 35 ;  104 36 i 11  105 36 =  106 36 i 8  107 36 i 9  108 36 v -  109 36 ;  110 37 i 12  111 37 =  112 37 i 8  113 37 i 9  114 37 v \*  115 37 ;  116 38 i 13  117 38 =  118 38 i 8  119 38 i 9  120 38 v / | 121 38 ;  122 39 i 14  123 39 =  124 39 i 8  125 39 i 9  126 39 v %  127 39 ;  128 40 i 15  129 40 =  130 40 l 21  131 40 ;  132 41 i 16  133 41 =  134 41 i 17  135 41 @ -3 z  136 41 1 -1 z  137 41 # -1  138 41 # -1  139 42 o  140 42 (  141 42 i 10  142 42 )  143 42 ;  144 43 o  145 43 (  146 43 i 11  147 43 )  148 43 ;  149 44 o  150 44 (  151 44 i 12  152 44 )  153 44 ;  154 45 o  155 45 (  156 45 i 13  157 45 )  158 45 ;  159 46 o  160 46 (  161 46 i 14  162 46 )  163 46 ;  164 47 o  165 47 (  166 47 i 15  167 47 )  168 47 ;  169 48 o  170 48 (  171 48 i 16  172 48 )  173 48 ;  174 50 c  175 50 (  176 50 i 16  177 50 y >  178 50 l 22  179 50 )  180 50 ? ?  181 51 {  182 52 o  183 52 (  184 52 l 23  185 52 )  186 52 ;  187 53 }  188 55 j  189 55 {  190 56 i 16  191 56 =  192 56 i 16  193 56 l 24  194 56 v \*  195 56 ;  196 57 o  197 57 (  198 57 i 16  199 57 )  200 57 ;  201 58 }  202 60 i 10  203 60 =  204 60 l 25  205 60 l 26  206 60 l 27  207 60 v %  208 60 l 28  209 60 l 29  210 60 v +  211 60 v \*  212 60 v -  213 60 ;  214 1 # -1  215 1 # -1  216 1 # -1  217 1 # -1  218 61 o  219 61 (  220 61 i 10  221 61 )  222 61 ;  223 63 i 17  224 63 =  225 63 l 30  226 63 ;  227 64 i 10  228 64 =  229 64 i 17  230 64 @ -2 z  231 64 1 -1 z  232 64 l 31  233 64 v \*  234 64 # -1  235 64 # -1  236 65 o  237 65 (  238 65 i 10  239 65 )  240 65 ;  241 67 i 17  242 67 =  243 67 i 8  244 67 i 9  245 67 @ 1  246 67 2 -1  247 67 ;  248 67 # -1  249 67 # -1  250 68 o  251 68 (  252 68 i 17  253 68 )  254 68 ;  255 69 r  256 69 l 32  257 69 ;  258 70 }  259 70 ; |

Листинг 4 Соответствие лексем исходному коду программы

|  |  |
| --- | --- |
| ===========LEX TABLE mini==============  tfi(ti,ti)  {  nti;  i=i;  c(i=i)?  {  i=l;  }  j{  i=l;  }  ri;  };  s  {  nti;  nti;  nti;  nti;  nti;  nti;  nti;  nti;  nti;  nti; | i=l;  i=l;  i=l;  i=ivi;  i=ivi;  i=ivi;  i=ivi;  i=ivi;  i=l;  i=i(i);  o(i);  o(i);  o(i);  o(i);  o(i);  o(i);  o(i);  c(iyl)?  {  o(l);  }  j{  i=ivl;  o(i);  }  i=lv(lvl)v(lvl);  o(i);  i=l;  i=i(i)vl;  o(i);  i=i(i,i);  o(i);  rl;  }; |

# 

# Приложение Д

Листинг 1 Результат генерации кода контрольного примера в Ассемблере

|  |  |
| --- | --- |
| .586  .model flat, stdcall  includelib kernel32.lib  includelib libucrt.lib  includelib mylib.lib  ExitProcess PROTO : DWORD  outtxt PROTO : DWORD  outlit PROTO : SDWORD  sleep PROTO  textlenght PROTO : DWORD  texttolit PROTO : DWORD  copytxt PROTO : DWORD,:DWORD  .stack 4096  .const  mesdivbyzero byte 'Divide by zero',0  T0 byte 'Yes', 0  T1 byte 'No', 0  T2 byte '3', 0  L0 sdword 13  L1 sdword 7  L2 sdword 15  L3 sdword 5  T3 byte 'Yes', 0  L4 sdword 2  L5 sdword 13  L6 sdword 21  L7 sdword 4  L8 sdword 2  L9 sdword 1  T4 byte '15', 0  L10 sdword 2  L11 sdword 0  .data  buf byte 255 dup(0)  buffer00000 sdword ?  res01000 byte 255 dup(0)  T0T byte 255 dup(0)  T1T byte 255 dup(0)  f04000 sdword ?  s04000 sdword ?  summa04000 sdword ?  raznost04000 sdword ?  proizv04000 sdword ?  delen04000 sdword ?  ostdelen04000 sdword ?  octo04000 sdword ?  oftext04000 sdword ?  tonumb04000 byte 255 dup(0)  T2T byte 255 dup(0)  T3T byte 255 dup(0)  T4T byte 255 dup(0)  .code  proc\_iftest proc, f01000 : dword, s01000 : dword  push s01000  pop f01000  push f01000  push s01000  pop eax  pop ebx  cmp eax,ebx  jne CKECKNOT1  push offset T0  push offset res01000  call copytxt  jmp OUTCHECK1  CKECKNOT1 :  push offset T1  push offset res01000  call copytxt  OUTCHECK1 :  mov eax, offset res01000  jmp toend  divbyzero:  push offset mesdivbyzero  call outtxt  call sleep  call ExitProcess  toend:  ret  proc\_iftest endp  main proc  START :  push offset T2  push offset tonumb04000  call copytxt  push L0  pop f04000  push L1  pop s04000  push f04000  push s04000  pop eax  pop ebx  add eax, ebx  push eax  pop summa04000  push f04000  push s04000  pop ebx  pop eax  sub eax, ebx  push eax  pop raznost04000  push f04000  push s04000  pop eax  pop ebx  mul ebx  push eax  pop proizv04000  push f04000  push s04000 | pop ebx  pop eax  cmp ebx,0  je divbyzero  mov edx,0  idiv ebx  push eax  pop delen04000  push f04000  push s04000  pop ebx  pop eax  cmp ebx,0  je divbyzero  mov edx,0  idiv ebx  push edx  pop ostdelen04000  push L2  pop octo04000  push offset tonumb04000  pop edx  push offset tonumb04000  call texttolit  push eax  pop oftext04000  push summa04000  call outlit  push raznost04000  call outlit  push proizv04000  call outlit  push delen04000  call outlit  push ostdelen04000  call outlit  push octo04000  call outlit  push oftext04000  call outlit  push oftext04000  push L3  pop eax  pop ebx  cmp eax,ebx  jnl CKECKNOT2  push offset T3  call outtxt  jmp OUTCHECK2  CKECKNOT2 :  push oftext04000  push L4  pop eax  pop ebx  mul ebx  push eax  pop oftext04000  push oftext04000  call outlit  OUTCHECK2 :  push L5  push L6  push L7  pop ebx  pop eax  cmp ebx,0  je divbyzero  mov edx,0  idiv ebx  push edx  push L8  push L9  pop eax  pop ebx  add eax, ebx  push eax  pop eax  pop ebx  mul ebx  push eax  pop ebx  pop eax  sub eax, ebx  push eax  pop summa04000  push summa04000  call outlit  push offset T4  push offset tonumb04000  call copytxt  push offset tonumb04000  pop edx  push offset tonumb04000  call textlenght  push eax  push L10  pop eax  pop ebx  mul ebx  push eax  pop summa04000  push summa04000  call outlit  push f04000  push s04000  pop edx  pop edx  push s04000  push f04000  call proc\_iftest  push eax  push offset tonumb04000  call copytxt  push offset tonumb04000  call outtxt  mov eax,L11  push 0  jmp toend  divbyzero:  push offset mesdivbyzero  call outtxt  toend:  call sleep  call ExitProcess  main endp  end main |