МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

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

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

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора NEV-2023»

Выполнил студент Нехайчик Евгений Владимирович

(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

Содержание

[Введение 4](#_Toc153841088)

[1 Спецификация языка программирования 5](#_Toc153841089)

[1.1 Характеристика языка программирования 5](#_Toc153841090)

[1.2 Определение алфавита языка программирования 5](#_Toc153841091)

[1.3 Применяемые сепараторы 5](#_Toc153841092)

[1.4 Применяемые кодировки 6](#_Toc153841093)

[1.5 Типы данных 6](#_Toc153841094)

[1.6 Преобразование типов данных 7](#_Toc153841095)

[1.7 Идентификаторы 7](#_Toc153841096)

[1.8 Литералы 7](#_Toc153841097)

[1.9 Объявление данных 8](#_Toc153841098)

[1.10 Инициализация данных 8](#_Toc153841099)

[1.11 Инструкции языка 8](#_Toc153841100)

[1.12 Операции языка 9](#_Toc153841101)

[1.13 Выражения и их вычисление 10](#_Toc153841102)

[1.14 Конструкции языка 10](#_Toc153841103)

[1.15 Области видимости идентификаторов 11](#_Toc153841104)

[1.16 Семантические проверки 11](#_Toc153841105)

[1.17 Распределение оперативной памяти на этапе выполнения 12](#_Toc153841106)

[1.18 Стандартная библиотека и её состав 12](#_Toc153841107)

[1.19 Ввод и вывод данных 12](#_Toc153841108)

[1.20 Точка входа 13](#_Toc153841109)

[1.21 Препроцессор 13](#_Toc153841110)

[1.22 Соглашения о вызовах 13](#_Toc153841111)

[1.23 Объектный код 13](#_Toc153841112)

[1.24 Классификация сообщений транслятора 13](#_Toc153841113)

[1.25 Контрольный пример 14](#_Toc153841114)

[2 Структура транслятора 15](#_Toc153841115)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 15](#_Toc153841116)

[2.2 Перечень параметров транслятора 16](#_Toc153841117)

[2.3 Протоколы, формируемые транслятором 16](#_Toc153841118)

[3 Разработка лексического анализатора 18](#_Toc153841119)

[3.1 Структура лексического анализатора 18](#_Toc153841120)

[3.2 Входные и выходные данные лексического анализатора 19](#_Toc153841121)

[3.3 Параметры лексического анализатора 19](#_Toc153841122)

[3.4 Алгоритм лексического анализа 19](#_Toc153841123)

[3.5. Контроль входных символов 19](#_Toc153841124)

[3.6 Удаление избыточных символов 20](#_Toc153841125)

[3.7 Перечень ключевых слов 20](#_Toc153841126)

[3.8 Основные структуры данных 22](#_Toc153841127)

[3.9 Структура и перечень сообщений лексического анализатора 24](#_Toc153841128)

[3.10 Принцип обработки ошибок 24](#_Toc153841129)

[3.10 Контрольный пример 24](#_Toc153841130)

[4. Разработка синтаксического анализатора 25](#_Toc153841131)

[4.1 Структура синтаксического анализатора 25](#_Toc153841132)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 25](#_Toc153841133)

[4.3 Построение конечного магазинного автомата 27](#_Toc153841134)

[4.4 Основные структуры данных 28](#_Toc153841135)

[4.5 Описание алгоритма синтаксического разбора 28](#_Toc153841136)

[4.6 Параметры синтаксического анализатора и режимы его работы 29](#_Toc153841137)

[4.7 Структура и перечень сообщений синтаксического анализатора 29](#_Toc153841138)

[4.8 Принцип обработки ошибок 29](#_Toc153841139)

[4.9 Контрольный пример 29](#_Toc153841140)

[5 Разработка семантического анализатора 30](#_Toc153841141)

[5.1 Структура семантического анализатора 30](#_Toc153841142)

[5.2 Функции семантического анализатора 30](#_Toc153841143)

[5.3 Структура и перечень сообщений семантического анализатора 30](#_Toc153841144)

[5.4 Принцип обработки ошибок 31](#_Toc153841145)

[5.5 Контрольный пример 31](#_Toc153841146)

[6. Вычисление выражений 32](#_Toc153841147)

[6.1 Выражения, допускаемые языком 32](#_Toc153841148)

[6.2 Польская запись и принцип её построения 32](#_Toc153841149)

[6.3 Программная реализация обработки выражений 32](#_Toc153841150)

[6.4 Контрольный пример 33](#_Toc153841151)

[7. Генерация кода 34](#_Toc153841152)

[7.1 Структура генератора кода 34](#_Toc153841153)

[7.2 Представление типов данных в оперативной памяти 34](#_Toc153841154)

[7.3 Статическая библиотека 35](#_Toc153841155)

[7.4 Особенности алгоритма генерации кода 35](#_Toc153841156)

[7.5 Параметры, управляющие генерацией кода 35](#_Toc153841157)

[7.6 Контрольный пример 36](#_Toc153841158)

[8. Тестирование транслятора 37](#_Toc153841159)

[8.1 Общие положения 37](#_Toc153841160)

[8.2 Результаты тестирования 37](#_Toc153841161)

[8.2.1 Тестирование проверки на допустимость символов 37](#_Toc153841162)

[8.2.2 Тестирование лексического анализатора 37](#_Toc153841163)

[8.2.3 Тестирование синтаксического анализатора 38](#_Toc153841164)

[8.2.4 Тестирование семантического анализатора 39](#_Toc153841165)

[Заключение 41](#_Toc153841166)

[Графический материал 42](#_Toc153841167)

[Список использованных источников 43](#_Toc153841168)

[Приложение А 44](#_Toc153841169)

[Приложение Б 46](#_Toc153841170)

[Приложение В 50](#_Toc153841171)

[Приложение Г 56](#_Toc153841172)

[Приложение Д 62](#_Toc153841173)

# Введение

В данном курсовом проекте поставлена задача разработки собственного языка программирования и транслятора для него. Название языка – NEV-2023. Написание транслятора будет осуществляться на языке C++, при этом код на языке NEV-2023 будет транслироваться в язык ассемблера.

Задание на курсовой проект можно разделить на следующие задачи:

1. Разработка спецификации языка NEV-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 NEV-2023 является процедурным, универсальным, строго типизированным, не объектно-ориентированным, компилируемым.

## 1.2 Определение алфавита языка программирования

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

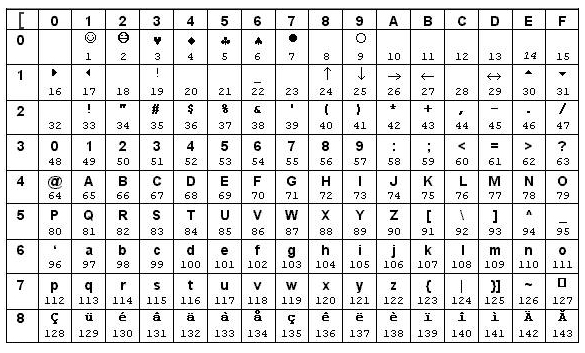


Рисунок 1.1 Алфавит входных символов

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 1.1.

Таблица 1.1 Символы-сепараторы

|  |  |
| --- | --- |
| Символ(ы) | Назначение |
| ‘пробел’ | Разделитель цепочек. Допускается везде кроме названий идентификаторов и ключевых слов |

Продолжение таблицы 1.1

|  |  |
| --- | --- |
| Символ(ы) | Назначение |
| **{** … **}** | Блок функции или условной конструкции/цикла |
| **(** … **)** | Блок фактических или формальных параметров функции, а также приоритет арифметических операций |
| **,** | Разделитель параметров функций |
| **#** | Символ, отделяющий условные конструкции/циклы |
| **+ - \* /** % | Арифметические операции |
| **> <** & ! | Логические операции (операции сравнения: больше, меньше, проверка на равенство, на неравенство), используемые в условии цикла/условной конструкции. |
| ; | Разделитель программных конструкций |
| **=** | Оператор присваивания |

## 1.4 Применяемые кодировки

Для написания программ язык NEV-2023 использует кодировку ASCII, содержащую английский алфавит, а также некоторые специальные символы, такие как ( ) , ; : # + - / \* % > < & !{}.

## 1.5 Типы данных

В языке NEV-2023 реализованы два фундаментальных типа данных: целочисленный и строковый. Описание типов приведено в таблице 1.2.

Таблица 1.2 Типы данных языка NEV-2023

|  |  |
| --- | --- |
| Тип данных | Характеристика |
| Целочисленный тип данных number | Фундаментальный тип данных. Используется для работы с числовыми значениями. В памяти занимает 4 байта. Максимальное значение: 2 147 483 647.  Минимальное значение: -2 147 483 648.  Инициализация по умолчанию: значение 0.  Поддерживаемые операции:  **+** (бинарный) – оператор сложения;  - (бинарный) – оператор вычитания;  **\*** (бинарный) – оператор умножения;  **/** (бинарный) – оператор деления нацело;  % (бинарный) – оператор остатка от деления;  **=** (бинарный) – оператор присваивания.  В качестве оператора цикла/условия поддерживаются операции:  **>** (бинарный) – оператор «больше»;  **<** (бинарный) – оператор «меньше»;  & (бинарный) – оператор проверки на равенство;  ! (бинарный) – оператор проверки на неравенство. |

Продолжение таблицы 1.2

|  |  |
| --- | --- |
| Тип данных | Характеристика |
| Строковый тип данных string | Фундаментальный тип данных. Используется для работы с символами, каждый из которых занимает 1 байт. Максимальное количество символов – 255.  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. |

## 1.6 Преобразование типов данных

Преобразование типов данных в языке NEV-2023 не предусмотрено.

## 1.7 Идентификаторы

Общее количество идентификаторов ограничено максимальным размером таблицы идентификаторов. Идентификаторы должны содержать только символы нижнего регистра латинского алфавита. Максимальная длина идентификатора равна восьми символам. Идентификаторы, объявленные внутри функционального блока, получают префикс, идентичный имени функции, внутри которой они объявлены. Префикс занимает 8 дополнительных символов. В случае превышения заданной длины, идентификаторы усекаются до длины, равной 16 символов (8 символов на имя идентификатора, 8 символов на префикс). Данные правила действуют для всех типов идентификаторов. Зарезервированные идентификаторы не предусмотрены. Идентификаторы не должны совпадать с ключевыми словами. Типы идентификаторов: имя переменной, имя функции, параметр функции. Имена идентификаторов-функций не должны совпадать с именами команд ассемблера (это не касается имён идентификаторов-переменных).

## 1.8 Литералы

С помощью литералов осуществляется инициализация переменных. Все литералы являются rvalue. Типы литералов языка NEV-2023 представлены в таблице 1.3.

Таблица 1.3 Литералы

|  |  |
| --- | --- |
| Литералы | Пояснение |
| Целочисленные литералы в десятичном представлении | Последовательность цифр 0…9 с предшествующим знаком минус или без него |
| Строковые литералы | Набор символов (от 1 до 255), заключённых в двойные кавычки |

Продолжение таблицы 1.3

|  |  |
| --- | --- |
| Целочисленные литералы в шестнадцатеричном представлении | Последовательность цифр 0…9 и символов A…F с предшествующим символом ‘q’ |

Ограничения на строковые литералы языка NEV-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. Допускается инициализация при объявлении.

Пример объявления числового типа с инициализацией:

new number num1 = -1;

new number num2 = 12;

Пример объявления переменной символьного типа с инициализацией:

new string str1= “hello world”;

Для объявления функций используется ключевое слово function, перед которым указывается тип функции (если функция возвращает значение), или ключевое слово void, если функция ничего не возвращает, а после – имя функции. Далее обязателен список параметров и тело функции.

## 1.10 Инициализация данных

При объявлении переменной допускается инициализация данных. При этом переменной будет присвоено значение литерала или идентификатора, стоящего справа от знака равенства. Объектами-инициализаторами могут быть только идентификаторы или литералы. При объявлении без инициализации предусмотрены значения по умолчанию: значение 0 для типа number и строка длины 0 (“”) для типа string.

## 1.11 Инструкции языка

Инструкции языка NEV-2023 представлены в таблице 1.4.

Таблица 1.4 ИнструкцииязыкаNEV-2023

|  |  |
| --- | --- |
| Инструкция | Реализация |
| Объявление переменной | **new** <тип данных> <идентификатор>; |

Продолжение таблицы 1.4

|  |  |
| --- | --- |
| Инструкция | Реализация |
| Объявление переменной с явной инициализацией | **new** <тип данных> <идентификатор> = <значение>;  Значение – инициализатор конкретного типа. Может быть только литералом или идентификатором |
| Возврат из функции с возвращающим или без возвращающего значений | Для функций, возвращающих значение:  **return** <идентификатор/литерал>;  Для функций, не возвращающих значение:  **return;** |
| Вывод данных | **print** <идентификатор/литерал>; |
| Вызов функции | <идентификатор функции> (<список параметров>);  Список параметров может быть пустым. |
| Перевод строки | **newline**; |
| 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 NEV-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 1.5.

Таблица 1.5 Операции языка NEV-2023

|  |  |
| --- | --- |
| Тип оператора | Оператор |
| Арифметические | 1. **+ –** сложение 2. - – вычитание 3. \* – умножение 4. **/** – деление нацело 5. % – остаток от деления 6. = – присваивание |
| Строковые | 1. **=** – присваивание |
| Логические | 1. **>** – больше  2. **<** – меньше  3. & – проверка на равенство  4. ! – проверка на неравенство |

## 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;
4. Допускается использовать в выражении вызов функции, вычисляющей и возвращающей значение.

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

## 1.14 Конструкции языка

Программа на языке NEV-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 1.6.

Таблица 1.6 Программные конструкции языка NEV-2023

|  |  |
| --- | --- |
| Конструкция | Реализация |
| Главная функция | **main**  **{**  …  **}** |
| Внешняя функция с возвращающим значением | <тип данных> **function** <идентификатор> **(**<тип> <идентификатор>, ...**)**  **{**…  **return** <идентификатор/литерал>;  **}** |
| Внешняя функция без возвращающего значения | **void** **function** <идентификатор> **(**<тип> <идентификатор>, ...**)**  **{**…  **return**;  **}** |
| Цикл | **criterion:** <идентификатор1> <оператор> <идентификатор2>  **#**  **cycle** **{** … **}**  **#**  Цикл (операторы внутри блока **cycle**) выполняется, пока истинно условие “<идентификатор1> <оператор> <идентификатор2>”, имеющее тот же смысл, что и в примере выше. |

Продолжение таблицы 1.6

|  |  |
| --- | --- |
| Конструкция | Реализация |

|  |  |
| --- | --- |
| Условная конструкция | **criterion:** <идентификатор1> <оператор> <идентификатор2>  **#**  **istrue** { … **}**  **isfalse** **{** … **}**  **#**  <идентификатор1>, <идентификатор2> - идентификаторы или литералы целочисленного типа (но не два литерала одновременно). <оператор> - один из операторов сравнения ( **> < & !** ), устанавливающий отношение между двумя операндами и организующий условие данной конструкции. При истинности условия выполняется код внутри блока **istrue**, иначе – код внутри блока **isfalse**. Любой из блоков **istrue**, **isfalse** может отсутствовать, но не оба блока одновременно. При отсутствии одного из блоков, в зависимости от истинности или ложности условия программа может как выполнить один из заявленных блоков, так и передать управление инструкции, следующей в коде за закрывающим условную конструкцию символом **‘#’**. |

## 1.15 Области видимости идентификаторов

Область видимости: сверху вниз (как и в С++). Переменные, объявленные в одной функции, недоступны в другой. Все объявления и операции с переменными происходят внутри какого-либо блока. Каждая переменная или параметр функции получают префикс – название функции, внутри которой они находятся.

Все идентификаторы являются локальными и обязаны быть объявленными внутри какой-либо функции. Глобальных переменных нет. Параметры видны только внутри функции, в которой объявлены.

## 1.16 Семантические проверки

В языке программирования NEV-2023 выполняются следующие семантические проверки:

1. Наличие функции **main** – точки входа в программу;
2. Единственность точки входа;
3. Переопределение идентификаторов;
4. Использование идентификаторов без их объявления;
5. Проверка соответствия типа функции и возвращаемого параметра;
6. Правильность передаваемых в функцию параметров: количество, типы;
7. Правильность строковых выражений;
8. Превышение размера строковых и числовых литералов;
9. Правильность составленного условия цикла/условного оператора.

## 1.17 Распределение оперативной памяти на этапе выполнения

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

## 1.18 Стандартная библиотека и её состав

В языке NEV-2023 присутствует стандартная библиотека, которая подключается автоматически при трансляции исходного кода в язык ассемблера. Содержимое библиотеки и описание функций представлено в таблице 1.8.

Таблица 1.8 Стандартная библиотека языка NEV-2023

|  |  |
| --- | --- |
| Функция | Описание |
| **string** strcopy(**string** str); | Строковая функция, выполняет копирование строки str. Возвращает скопированную строку. |
| **number** atoii(**string** str); | Целочисленная функция. Преобразует строку в число |
| **number** length (**string** str); | 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 **print**. Эти функции представлены в таблице 1.9.

Таблица 1.9 Дополнительные функции стандартной библиотеки

|  |  |
| --- | --- |
| Функция на языке С++ | Описание |
| void outnum(int value) | Функция для вывода в стандартный поток значения целочисленного идентификатора/литерала |
| void outstr(char\* line) | Функция для вывода в стандартный поток значения строкового идентификатора/литерала |

## 1.19 Ввод и вывод данных

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

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

## 1.20 Точка входа

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

## 1.21 Препроцессор

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

## 1.22 Соглашения о вызовах

В языке вызов функций происходит по соглашению о вызовах stdcall.

Особенности stdcall:

– все параметры функции передаются через стек;

– память высвобождает вызываемый код;

– занесение в стек параметров идёт справа налево.

## 1.23 Объектный код

Язык NEV-2023 транслируется в язык ассемблера, а затем – в объектный код.

## 1.24 Классификация сообщений транслятора

Генерируемые транслятором сообщения определяют степень его информативности, то есть сообщения транслятора должны давать максимально полную информацию о допущенной пользователем ошибке при написании программы. Сообщения транслятора приведены в таблице 1.10, а также в приложении А.

Таблица 1.10 Классификация ошибок

|  |  |
| --- | --- |
| Номера ошибок | Характеристика |
| 0 – 200 | Системные ошибки |
| 200 – 299 | Ошибки лексического анализа |
| 300 – 399 | Ошибки семантического анализа |
| 600 – 699 | Ошибки синтаксического анализа |
| 400-499, 700-999 | Зарезервированные коды ошибок |

## 1.25 Контрольный пример

Контрольный пример демонстрирует главные особенности языка NEV-2023: его фундаментальные типы, основные структуры, функции, использование функций статической библиотеки. Исходный код контрольного примера представлен в приложении А.

# 2 Структура транслятора

## 2.1 Компоненты транслятора, их назначение и принципы взаимодействия

В языке NEV-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. Структура транслятора языка NEV-2023 приведена на рисунке 1.



Рисунок 2.1 Структура транслятора языка программирования NEV-2023

Первая стадия работы компилятора называется лексическим анализом, а программа, её реализующая, – лексическим анализатором (сканером). На вход лексического анализатора подаётся последовательность символов входного языка. Он 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 2.1.

Таблица 2.1 Входные параметры транслятора языка NEV-2023

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

## 2.3 Протоколы, формируемые транслятором

В таблице 2.2 приведены протоколы, формируемые транслятором и их содержимое.

Таблица 2.2 Протоколы, формируемые транслятором языка NEV-2023

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |

Продолжение таблицы 2.2

|  |  |
| --- | --- |
| Файл журнала, заданный параметром "-log:" | Файл с протоколом работы транслятора языка программирования NEV-2023. Содержит таблицу лексем и таблицу идентификаторов, протокол работы синтаксического анализатора и дерево разбора, полученные на этапе лексического и синтаксического анализа, а также результат работы алгоритма преобразования выражений к польской записи. |
| 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.

# 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 и 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 (альтернативный способ, состоящий в модификации грамматики для включения «пустых» символов и комментариев в синтаксис, достаточно сложен для реализации);

− распознавание идентификаторов и ключевых слов;

− распознавание констант;

− распознавание разделителей и знаков операций.

Исходный код программы представлен в приложении А, структура лексического анализатора представлена на рисунке 3.1.



Рисунок 3.1 Структура лексического анализатора

## 3.2 Входные и выходные данные лексического анализатора

На вход лексического анализатора подаётся исходный код входного языка. Лексический анализатор выделяет в этой последовательности простейшие конструкции языка и производит предварительный разбор текста, преобразующий единый массив текстовых символов в массив токенов.

## 3.3 Параметры лексического анализатора

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

## 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.

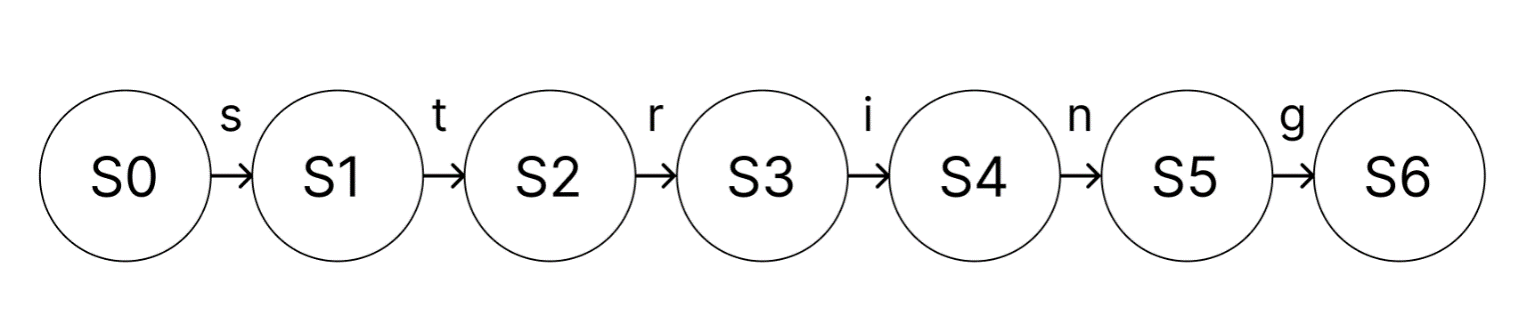
Распознавание цепочек основывается на работе конечных автоматов. Работу конечного автомата можно проиллюстрировать с помощью графа переходов. Пример графа для цепочки «**string**» представлен на рисунке 3.2, где S0 – начальное, а S6 – конечное состояние автомата.

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 3.1.



Рисунок 3.3. Таблица контроля входных символов

Таблица 3.1 Соответствие символов и их значений в таблице

|  |  |
| --- | --- |
| Значение в таблице входных символов | Символы |
| Разрешенный | T |
| Запрещенный | F |
| Игнорируемый | I |
| Литерал | Q |
| Сепаратор | S |
| Перевод строки | N |
| Пробел, табуляция | P |

## 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 является своего рода встречей символа-сепаратора;

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

## 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 3.2.

Таблица 3.2 Соответствие токенов и сепараторов с лексемами

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Пояснение |
| number, string | t | Названия типов данных языка. |
| Идентификатор | i | Длина идентификатора – 8 символов. |
| Литерал | l | Литерал любого доступного типа. |
| Шестнадцатеричный литерал | q | Литерал в шестнадцатеричном представлении. |
| function | f | Объявление функции. |
| script | p | Ключевое слово для скриптов – функций, не возвращающих значения. Указывается перед словом function. |
| return | e | Выход из функции/скрипта. |
| main | m | Главная функция. |
| new | n | Объявление переменной. |
| print | @ | Ввод данных. |
| criterion: | ? | Указывает начало цикла/условного оператора. |
| istrue | r | Истинная ветвь условного оператора. |
| isfalse | w | Ложная ветвь условного оператора. |
| cycle | c | Указывает на начало тела цикла. |
| newline | ^ | Оператор вывода символа перевода строки. |
| # | # | Разделение конструкций в цикле/условном операторе. |
| ; | ; | Разделение выражений. |
| , | , | Разделение параметров функций. |
| [ | [ | Начало блока/тела функции. |
| ] | ] | Закрытие блока/тела функции. |
| }  { | }  { | Знаки сдвиговых операций. |
| ( | ( | Передача параметров в функцию, приоритет операций. |
| ) | ) | Закрытие блока для передачи параметров, приоритет операций. |
| = | = | Знак присваивания. |
| +  -  \*  /  % | +  -  \*  /  % | Знаки операций. |
| >  <  &  ! | >  <  &  ! | Знаки логических операторов |

Пример реализации таблицы лексем представлен в приложении Б.

Каждому выражению соответствует детерминированный конечный автомат, по которому происходит разбор данного выражения. Структура конечного изображена на рисунке 3.4.

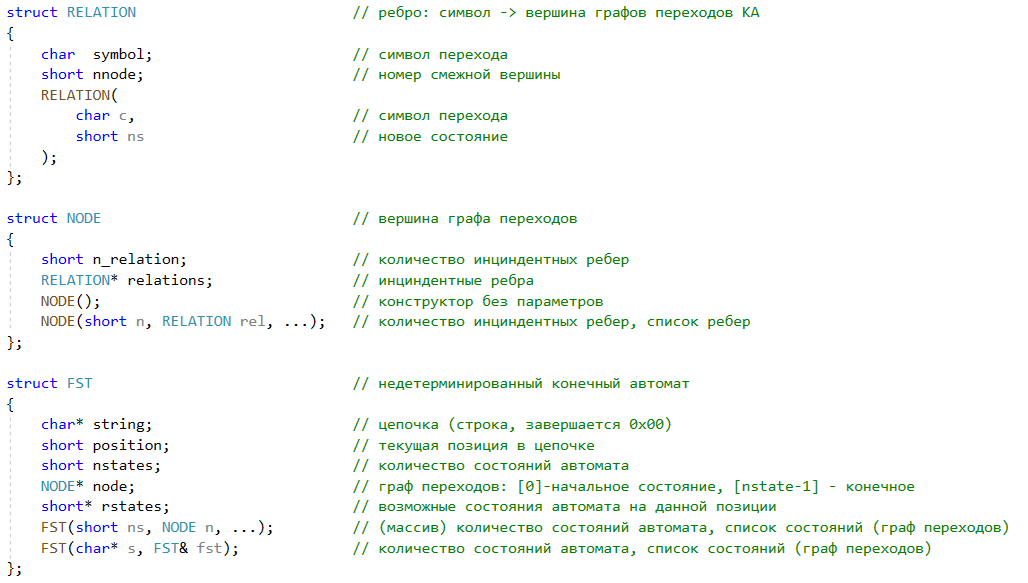


Рисунок 3.4 Структура конечного автомата

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. Пример графа перехода конечного автомата изображен на рисунке 3.5.

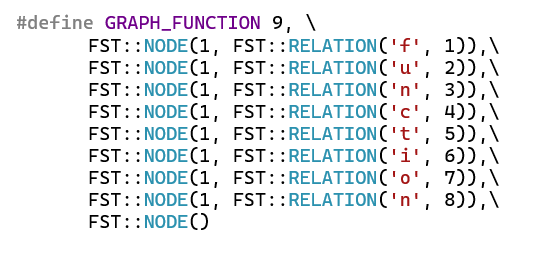


Рисунок 3.5 Пример реализации графа конечного автомата для токена criterion

## 3.8 Основные структуры данных

Основными структурами данных лексического анализатора являются таблица лексем и таблица идентификаторов. Таблица лексем содержит номер лексемы, лексему (lexema), полученную при разборе, номер строки в исходном коде (sn), и номер в таблице идентификаторов, если лексема является идентификаторов (idxTI). Код C++ со структурой таблицы лексем представлен на рисунке 3.6

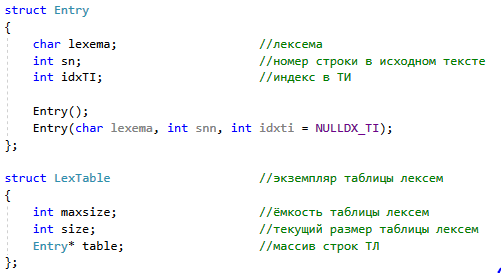


Рисунок 3.6 Структура таблицы лексем

Таблица идентификаторов содержит имя идентификатора (id), номер в таблице лексем (idxfirstLE), тип данных (iddatatype), тип идентификатора (idtype) и значение (или параметры функций) (value). Код C++ со структурой таблицы идентификаторов представлен на рисунке 3.7.

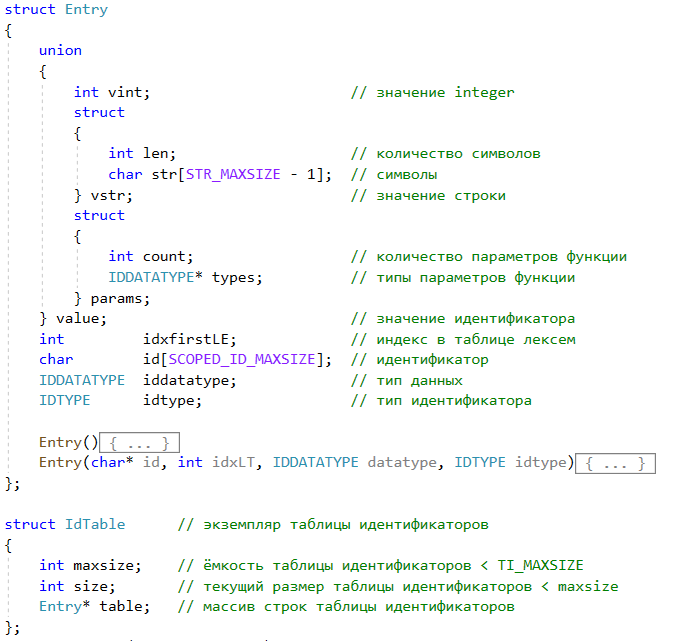


Рисунок 3.7 Структура таблицы идентификаторов

## 3.9 Структура и перечень сообщений лексического анализатора

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. Если в процессе анализа находятся более трёх ошибок, то анализ останавливается.

## 3.10 Принцип обработки ошибок

Для обработки ошибок лексический анализатор использует таблицу с сообщениями. Структура сообщений содержит информацию о номере сообщения и номер строки, 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 3.8.

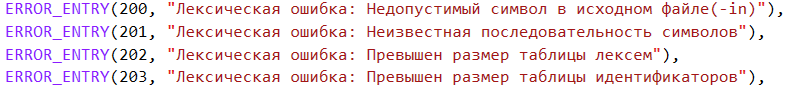


Рисунок 3.8 Сообщения лексического анализатора

## 3.10 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 NEV-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 4.1.

Таблица 4.1 Таблица правил переходов нетерминальных символов

|  |  |  |
| --- | --- | --- |
| Символ | Правила | Какие правила порождает |
| S | S->tfiPTS  S->pfiPGS  S->m[K] | Стартовые правила, описывающее общую структуру программы |
| P | P->(E)  P->() | Правила для параметров объявляемых функций |
| T | T->[eV;]  T->[KeV;] | Правила для тела функций |
| G | G->[e;]  G->[Ke;] | Правила для тела скриптов |
| E | E->ti,E  E->ti | Правила для списка параметров функции |
| F | F->(N)  F->() | Правила для вызова функций (в т.ч. и в выражениях) |
| N | N->i  N>l  N->l,N  N->I,N | Правила для параметров вызываемых функций |
| R | R->rY#  R>wY#  R>cY#  R->rYwY#  R->wYrY# | Правила составления цикла/условного оператора |
| Z | Z->iLi  Z->iLl  Z->lli | Правила для условия цикла/условного оператора |
| L | L-><  L->>  L->! | Правила для логических операторов |
| A | A->+  A->-  A->\*  A->/  A->{  A->} | Правила для арифметических и сдвиговых операторов |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Символ | Правила | Какие правила порождает |
| V | V->l  V->i  V->q | Правила для простых выражений |
| Y | Y->[X] | Правила для тела цикла/условного выражения |
| W | W->l  W->i  W->(W)  W->(W)AW  W->iF  W->iAW  W->lAW  W->iFAW | Правила для сложных выражений |
| K | K->nti=V;K  K->nti;K  K->i=W;K  K->oV;K  K->^;K  K->&Z#RK  K->iF;K  K->nti=V;  K->nti;  K->i=W;  K->oV;  K->^;  K->&Z#R  K->iF; | Программные конструкции |
| X | X->i=W;X  X->oV;X  X->^;X  X->iF;X  X->i=W;  X->oV;  X->^;  X->iF; | Программные конструкции внутри цикла/условного оператора |

## 4.3 Построение конечного магазинного автомата

Конечный автомат с магазинной памятью представляет собой семерку. Подробное описание компонентов магазинного автомата представлено в таблице 4.2.

Таблица 4.2 – 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 |
|  | 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 NEV-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 лексического 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 сообщений синтаксического анализатора представлен на рисунке 4.3.

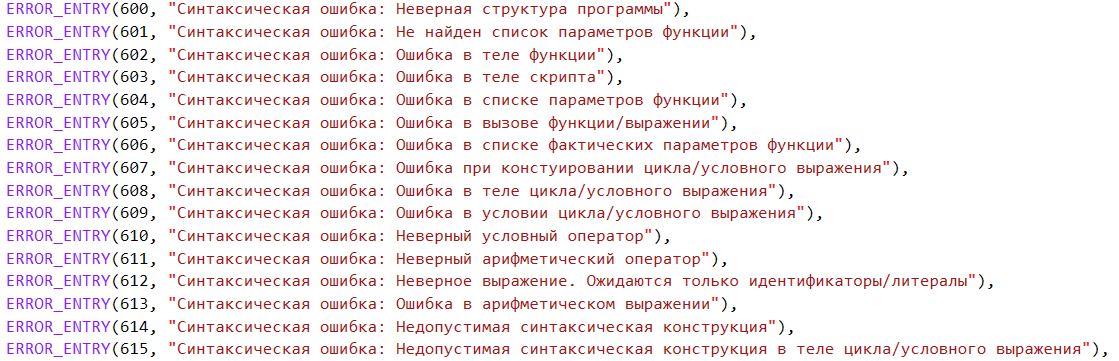


Рисунок 4.3 Сообщения синтаксического анализатора

## 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). Если в процессе анализа находятся более трёх ошибок, то анализ останавливается.

## 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. Некоторые проверки (такие как проверка на единственность точки входа, проверка на предварительное объявление переменной) осуществляются в процессе лексического анализа. 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

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

## 5.3 Структура и перечень сообщений семантического анализатора

Сообщения, формируемые семантическим анализатором, представлены на рисунке 5.2.

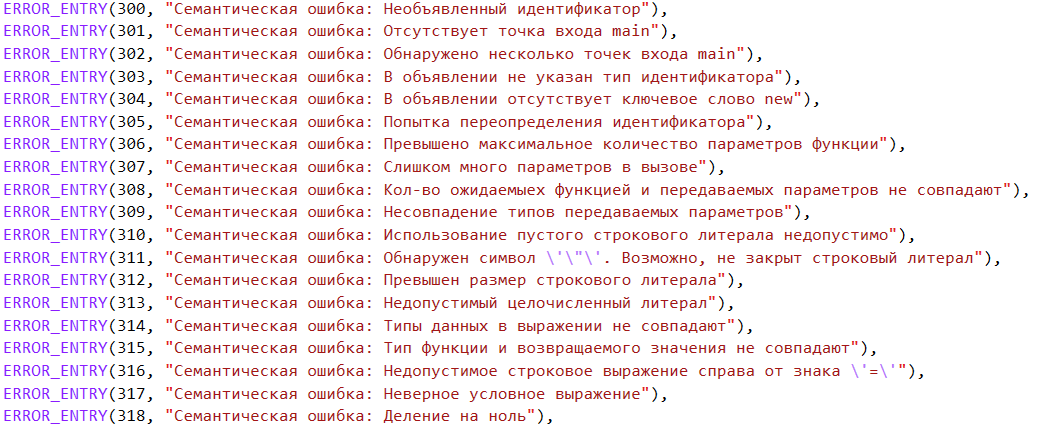


Рисунок 5.2 Перечень сообщений семантического анализатора

## 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. Анализ останавливается после того, как будут найдены все ошибки.

## 5.5 Контрольный пример

Соответствие примеров некоторых ошибок в исходном коде и диагностических сообщений об ошибках приведено в таблице 5.1.

Таблица 5.1. Примеры диагностики ошибок

|  |  |
| --- | --- |
| Исходный код | Текст сообщения |
| main  {  number x = 9;  print x;  } | Ошибка №304: Семантическая ошибка: В объявлении отсутствует ключевое слово new Строка: 2 |
| main  {  new number x = 9;  new string y =x;  } | Ошибка №314: Семантическая ошибка: Типы данных в выражении не совпадают Строка: 3 |
| main  {  new number x = 9;  }  main  {  new string y = "qwerty";  } | Ошибка №302: Семантическая ошибка: Обнаружено несколько точек входа main Строка: 0 |

В таблице представленные не все варианты ошибок, которые могут быть обнаружены на этапе семантического анализатора.

# 6. Вычисление выражений

## 6.1 Выражения, допускаемые языком

В языке NEV-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 |
| ( ) | 3 |
| \* | 2 |
| / | 2 |
| % | 2 |
| + | 1 |
| - | 1 |

## 6.2 Польская запись и принцип её построения

Все выражения языка NEV-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;
* отрывающая скобка помещается в стек;
* закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются.

## 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.

Таблица 6.2. Преобразование выражений к ПОЛИЗ

|  |  |
| --- | --- |
| Выражение | Обратная польская запись для выражения |
| i[2]=(((l[3]+l[4])-i[0])\*l[5])/l[6]; | i[2]=l[3]l[4]+i[0]-l[5]\*l[6]/ |
| i[23]=(i[23]+l[26])\*l[26] | i[23]=i[23]l[26]+l[26]\* |
| i[3]=(((l[4]+l[5])-i[0])\*l[6]) | i[3]=l[4]l[5]+i[0]-l[6]\* |

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.

# 7. Генерация кода

## 7.1 Структура генератора кода

В языке NEV-2023 генерация кода является заключительным этапом трансляции. Генератор принимает на вход таблицы лексем и идентификаторов, полученные в результате лексического анализа. 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 NEV-2023 представлена на рисунке 7.1.

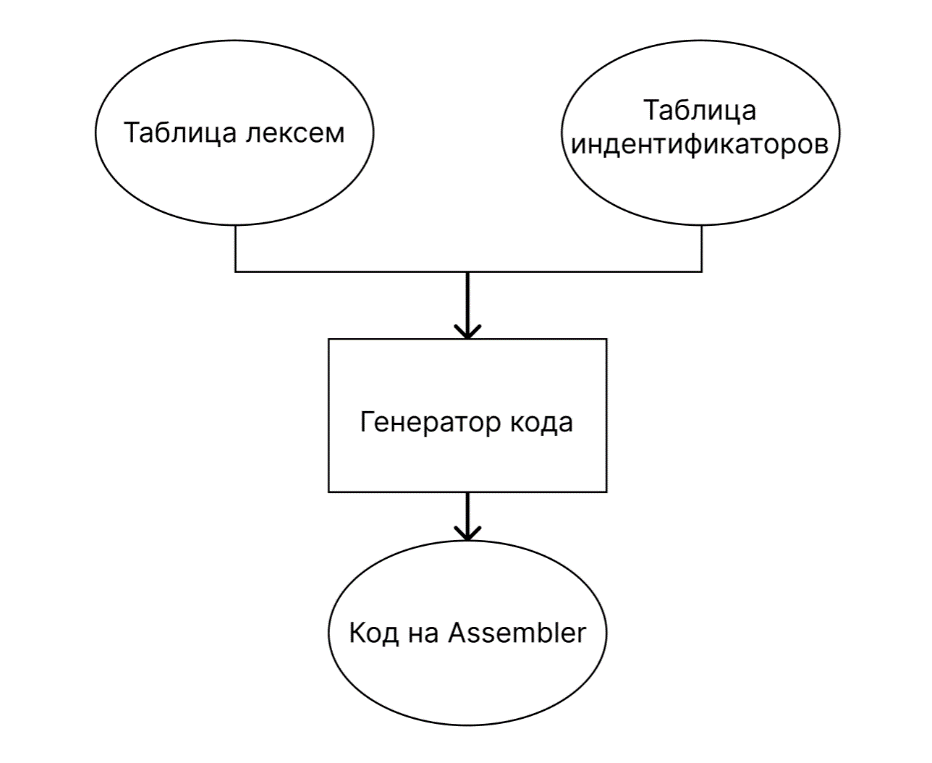


Рисунок 7.1 – Структура генератора кода

## 7.2 Представление типов данных в оперативной памяти

Элементы таблицы идентификаторов расположены сегментах .data и. const языка ассемблера. Соответствия между типами данных идентификаторов на языке NEV-2023 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка NEV-2023 и языка ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке NEV-2023 | Тип идентификатора на языке ассемблера | Пояснение |
| number | dword | Хранит целочисленный тип данных. |
| string | dword | Хранит указатель на начало строки. Строка должна завешаться нулевым символом. |

## 7.3 Статическая библиотека

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

Таблица 7.3 – Функции статической библиотеки

|  |  |
| --- | --- |
| Функция | Назначение |
| void outlstr (char\* str) | Вывод на консоль строки string |
| void outnum (int num) | Вывод на консоль целочисленной переменной number |
| int length (char\* buffer, char\* str) | Вычисление длины строки |
| char\* strcopy (char\* str) | Копирование строки str |
| int atoii (char\* ptr) | Преобразование строки в число |

## 7.4 Особенности алгоритма генерации кода

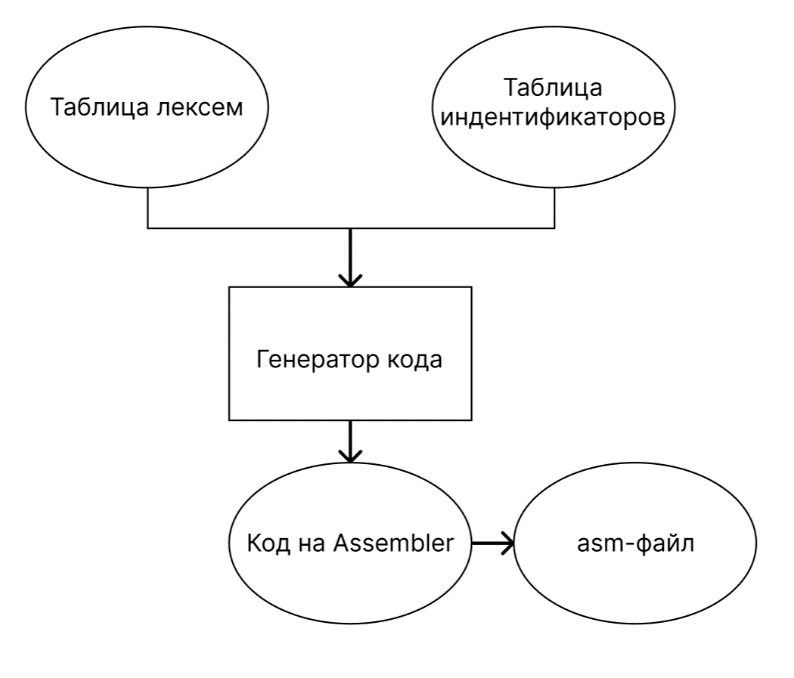
В языке NEV-2023 генерация кода строится на основе таблиц лексем и идентификаторов. Общая схема работы генератора кода представлена на рисунке 7.2.

Рисунок 7.2 Структура генератора кода

## 7.5 Параметры, управляющие генерацией кода

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

## 7.6 Контрольный пример

Результат работы контрольного примера приведён на рисунке 7.2.

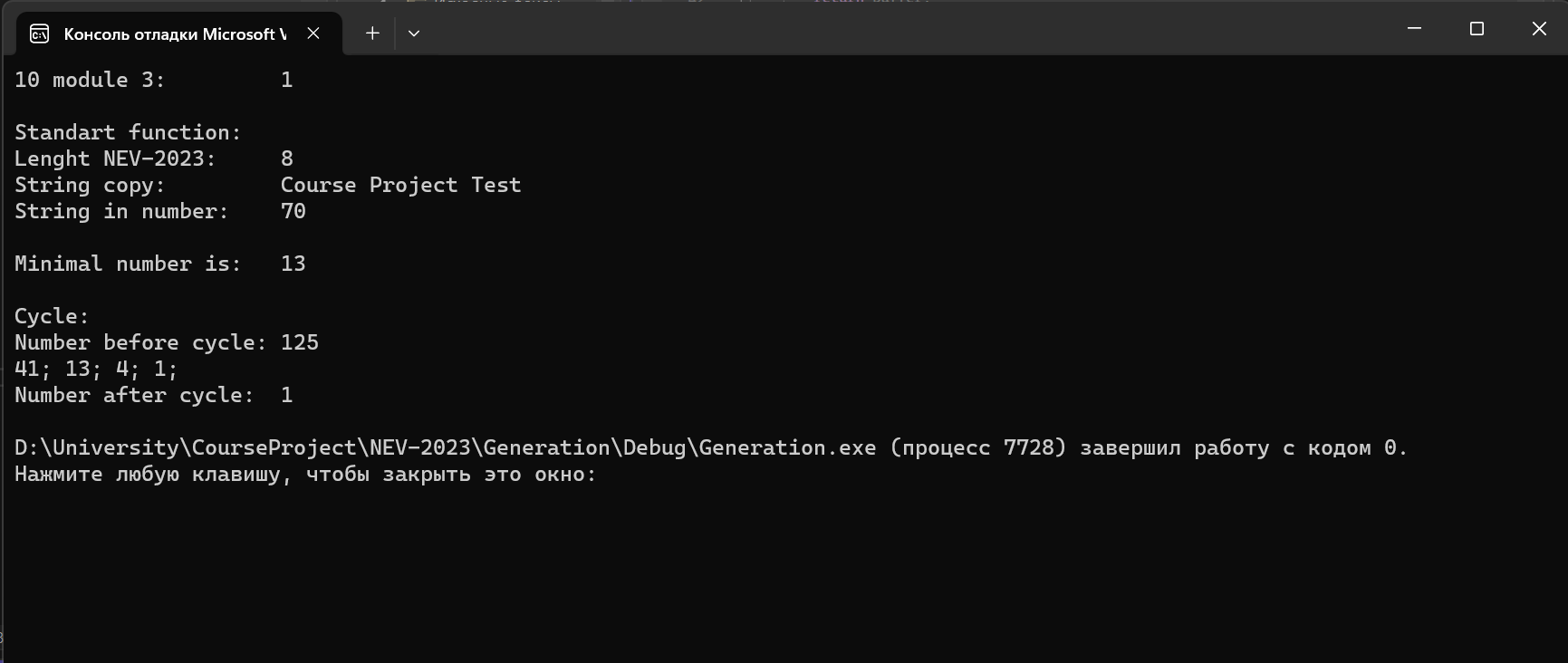


Рисунок 7.2 Результат работы программы на языке NEV-2023

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

# 8. Тестирование транслятора

## 8.1 Общие положения

**Общие принципы тестирования:**

1.Полнота тестирования: Тестирование должно охватывать все аспекты программы, включая различные пути выполнения, граничные условия и негативные сценарии.

2.Независимость тестирования: Тесты должны быть независимыми друг от друга, чтобы результат одного теста не влиял на выполнение другого. Это обеспечивает более точную оценку качества программы.

3.Воспроизводимость тестирования: Тесты должны быть воспроизводимыми, чтобы можно было повторно запустить тесты и получить те же самые результаты.

**Действия компилятора при обнаружении ошибки:**

1.Выдача сообщения об ошибке: Компилятор выводит сообщение с описанием обнаруженной ошибки, указывая на конкретное место в исходном коде.

2.Прекращение компиляции: В случае обнаружения ошибки, компилятор может прекратить процесс компиляции, чтобы предотвратить создание исполняемого файла с потенциальными проблемами.

**Протоколы вывода результатов тестирования:**

Протоколы вывода результатов тестирования записываются в файле формата log.

## 8.2 Результаты тестирования

## 8.2.1 Тестирование проверки на допустимость символов

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

Таблица 8.1 - Тестирование проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main {ё} | Ошибка №200: Лексическая ошибка: Недопустимый символ в исходном файле(-in) Строка: 2 Позиция в строке: 2 |

## 8.2.2 Тестирование лексического анализатора

На этапе лексического анализа в языке NEV-2023 могут возникнуть ошибки, описанные в пункте 3.9. Результаты тестирования лексического анализатора показаны в таблице 8.2.

Таблица 8.2 - Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  {new number w11;} | Ошибка №201: Лексическая ошибка: Неизвестная последовательность символов Строка: 3 |

## 8.2.3 Тестирование синтаксического анализатора

На этапе синтаксического анализа в языке NEV-2023 могут возникнуть ошибки, описанные в пункте 4.7. Результаты тестирования синтаксического анализатора показаны в таблице 8.3.

Таблица 8.3 - Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main new number x; } | Ошибка №600: строка 1, Синтаксическая ошибка: Неверная структура программы |
| string function fi({}  main{} | Ошибка №601: строка 1, Синтаксическая ошибка: Не найден список параметров функции |
| string function fi()  {newline; print}  main{} | Ошибка №602: строка 1, Синтаксическая ошибка: Ошибка в теле функции |
| script function fi()  {newline print;}  main{} | Ошибка №603: строка 1, Синтаксическая ошибка: Ошибка в теле скрипта |
| script function fi(number number){}  main{} | Ошибка №604: строка 1, Синтаксическая ошибка: Ошибка в списке параметров функции |
| string function fi(number x){return 3;}  main {newline;fi(5,5; newline;} | Ошибка №605: строка 2, Синтаксическая ошибка: Ошибка в вызове функции/выражении |
| string function fi(number x){return 3;}  main {newline;fi(5,5,5 5);} | Ошибка №606: строка 2, Синтаксическая ошибка: Ошибка в списке фактических параметров функции |
| main {new number x; criterion: x > 2 # cycle #} | Ошибка №607: строка 1, Синтаксическая ошибка: Ошибка при конструировании цикла/условного выражения |
| main {new number x; criterion: x > 2 # cycle #} | Ошибка №608: строка 1, Синтаксическая ошибка: Ошибка в теле цикла/условного выражения |
| main {criterion: 1 = 2 #} | Ошибка №609: строка 1, Синтаксическая ошибка: Ошибка в условии цикла/условного выражения |
| main {criterion: 1 = 2 #} | Ошибка №610: строка 1, Синтаксическая ошибка: Неверный условный оператор |

Продолжение таблицы 8.3

|  |  |
| --- | --- |
| main {new number x; x = x ! x;} | Ошибка №611: строка 1, Синтаксическая ошибка: Неверный арифметический оператор |
| main {new number x; print new;} | Ошибка №612: строка 1, Синтаксическая ошибка: Неверное выражение. Ожидаются только идентификаторы/литералы |
| main {new number x; x = 1 ++;} | Ошибка №613: строка 1, Синтаксическая ошибка: Ошибка в арифметическом выражении |
| main {newline; 4;} | Ошибка №614: строка 1, Синтаксическая ошибка: Недопустимая синтаксическая конструкция |
| main {new number a; criterion: a < 3 # istrue {newline; 3;} #} | Ошибка №615: строка 1, Синтаксическая ошибка: Недопустимая синтаксическая конструкция в теле цикла/условного выражения |

## 8.2.4 Тестирование семантического анализатора

Семантический анализ в языке NEV-2023 содержит множество проверок по семантическим правилам, описанным в пункте 1.16.

Таблица 8.4 - Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main { a = 1 } | Ошибка №300: Семантическая ошибка: Необъявленный идентификатор. Строка: 1 |
| string function fi(){} | Ошибка №301: Семантическая ошибка: Отсутствует точка входа main. Строка: 0 |
| main {}  main{} | Ошибка №302: Семантическая ошибка: Обнаружено несколько точек входа main. Строка: 0 |
| main{a = 1;} | Ошибка №304: Семантическая ошибка: В объявлении отсутствует ключевое слово new. Строка: 1 |
| main{new number t;  new string t;} | Ошибка №305: Семантическая ошибка: Попытка переопределения идентификатора. Строка: 3 |
| script function fi(){}  main{fi(“a”,”b”,”c”,”d”} | Ошибка №307: Семантическая ошибка: Слишком много параметров в вызове. Строка: 1 |
| string function fi(string x, string y, string z, string s)  main{} | Ошибка №306: Семантическая ошибка: Превышено максимальное количество параметров функции. Строка: 1 |
| string function fi(string x){return "a";}  main{fi("a", "b");} | Ошибка №308: Семантическая ошибка: Кол-во ожидаемых функцией и передаваемых параметров не совпадают. Строка: 2 |
| string function fi(string x)[{return "a";}  main{fi("a", "b");} | Ошибка №309: Семантическая ошибка: Несовпадение типов передаваемых параметров Строка: 2 |

Продолжение таблицы 8.4

|  |  |
| --- | --- |
| main{new string x="";} | Ошибка №310: Семантическая ошибка: Использование пустого строкового литерала недопустимо. Строка: 1 |
| main{new string x=";} | Ошибка №311: Семантическая ошибка: Обнаружен символ '"'. Строка: 1 |
| main{new number x=99999999999999999;} | Ошибка №313: Семантическая ошибка: Недопустимый целочисленный литерал. Строка: 1 |
| main{new number x; x = 5 + "abc";} | Ошибка №314: Семантическая ошибка: Типы данных в выражении не совпадают. Строка: 1 |
| string function fi(){return 5;}  main{newline;} | Ошибка №315: Семантическая ошибка: Тип функции и возвращаемого значения не совпадают. Строка: 1 |
| main{new string x; x = "abc" + "d";} | Ошибка №316: Семантическая ошибка: Недопустимое строковое выражение справа от знака '='. Строка: 1 |
| main  {criterion: "string"& 6#  istrue{print "string";}} | Ошибка №317: Семантическая ошибка: Неверное условное выражение. Строка: 1 |
| main{new number a =5;  a = a/0; print a;} | Ошибка №318: Семантическая ошибка: Деление на ноль. Строка: 4 |

Итоги тестирования семантического анализатора на корректное обнаружение семантических ошибок приведены в таблице 8.4.

# Заключение

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

1. Сформулирована спецификация языка NEV-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.

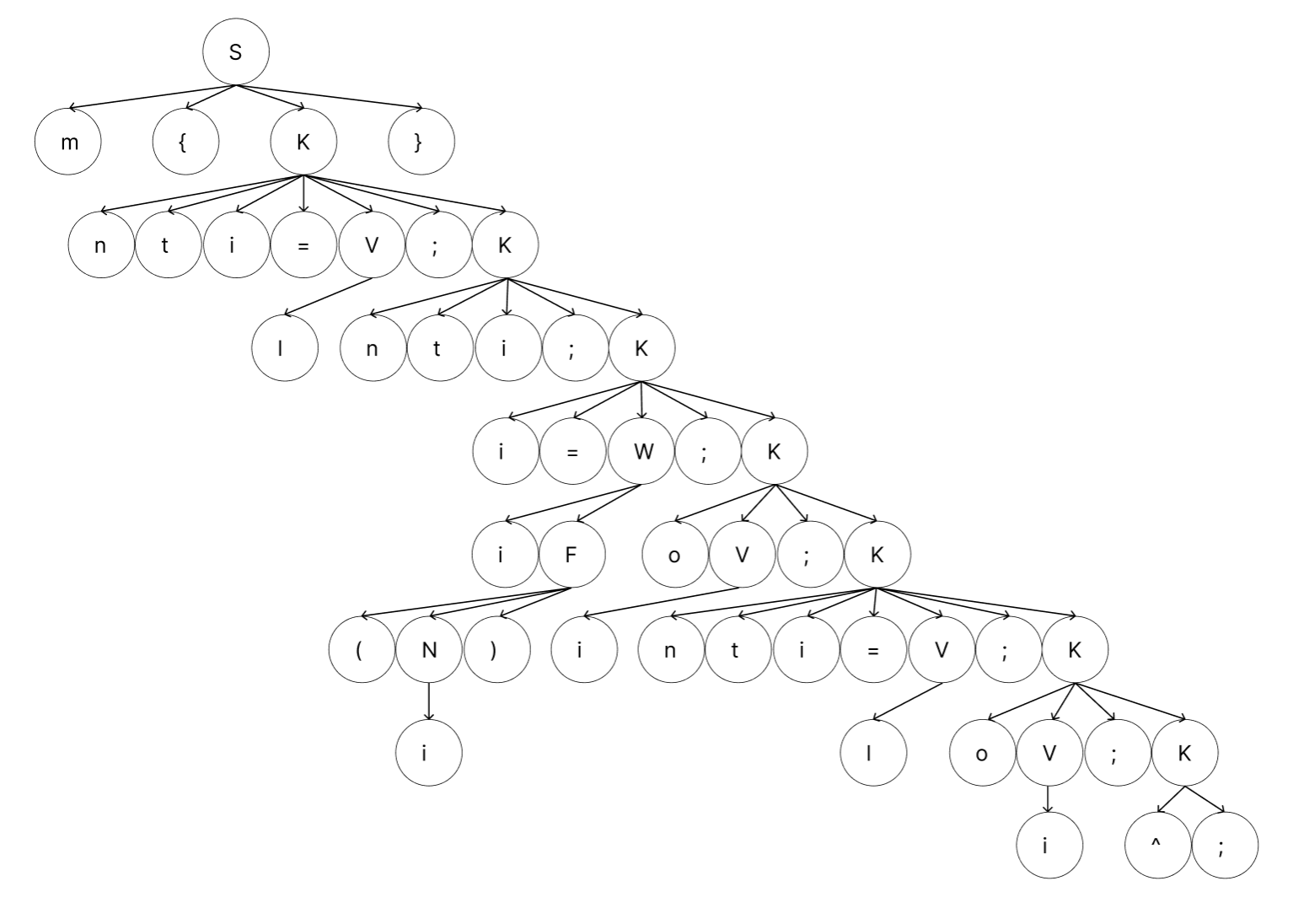
Окончательная версия языка NEV-2023 включает:

1. 2 типа данных;
2. Поддержка операторов вывода и перевода строки;
3. Возможность вызова функций стандартной библиотеки;
4. Наличие 5 арифметических операторов для вычисления выражений;
5. Поддержка функций, скриптов, операторов цикла 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 NEV-2023

number function min(number x, number y)

{

new number res;

criterion: x < y

#

istrue {res = x;}

isfalse {res = y;}

#

print "Minimal number is: ";

print res;

newline; newline;

return res;

}

number function div(number num)

{

num = num / 3;

print num;

print "; ";

return num;

}

void function standf (string a, string b, string c)

{

print "Standart function: ";

newline;

new number k;

k = lenght(a);

print "Lenght NEV-2023: ";

print k;

newline;

new string str;

str = strcopy(b);

print "String copy: ";

print str;

newline;

new number num;

num = atoii(c);

print "String in number: ";

print num;

newline; newline;

return;

}

main

{

new number x = 10;

new number y = 3;

x = x % y;

print "10 module 3: ";

print x;

newline; newline;

new string strx = "NEV-2023";

new string stry = "Course Project Test";

new string strz = "70";

standf(strx, stry, strz);

new number minone = 21;

new number mintwo = 13;

new number resmin;

resmin = min (minone,mintwo);

print "Cycle: ";

newline;

new number ab = 125;

print "Number before cycle: ";

print ab;

newline;

criterion: ab > 1

#

cycle

{

ab = div(ab);

}

#

newline;

print "Number after cycle: ";

print ab;

newline;

}

Листинг 2 Таблица ошибок

ERROR\_ENTRY(0, "Системная ошибка: Недопустимый код ошибки"),

ERROR\_ENTRY(1, "Системная ошибка: Системный сбой"),

ERROR\_ENTRY\_NODEF(2), ERROR\_ENTRY\_NODEF(3), ERROR\_ENTRY\_NODEF(4), ERROR\_ENTRY\_NODEF(5),

ERROR\_ENTRY\_NODEF(6), ERROR\_ENTRY\_NODEF(7), ERROR\_ENTRY\_NODEF(8), ERROR\_ENTRY\_NODEF(9),

ERROR\_ENTRY\_NODEF10(10), ERROR\_ENTRY\_NODEF10(20), ERROR\_ENTRY\_NODEF10(30), ERROR\_ENTRY\_NODEF10(40),

ERROR\_ENTRY\_NODEF10(50), ERROR\_ENTRY\_NODEF10(60), ERROR\_ENTRY\_NODEF10(70), ERROR\_ENTRY\_NODEF10(80), ERROR\_ENTRY\_NODEF10(90),

ERROR\_ENTRY(100, "Системная ошибка: Параметр -in должен быть задан"),

ERROR\_ENTRY(101, "Системная ошибка: Превышена длина входного параметра"),

ERROR\_ENTRY(102, "Системная ошибка: Ошибка при открытии файла с исходным кодом(-in)"),

ERROR\_ENTRY(103, "Системная ошибка: Ошибка при создании файла протокола(-log)"),

ERROR\_ENTRY\_NODEF(104), ERROR\_ENTRY\_NODEF(105), ERROR\_ENTRY\_NODEF(106), ERROR\_ENTRY\_NODEF(107), ERROR\_ENTRY\_NODEF(108), ERROR\_ENTRY\_NODEF(109),

ERROR\_ENTRY\_NODEF10(110), ERROR\_ENTRY\_NODEF10(120), ERROR\_ENTRY\_NODEF10(130), ERROR\_ENTRY\_NODEF10(140),

ERROR\_ENTRY\_NODEF10(150), ERROR\_ENTRY\_NODEF10(160), ERROR\_ENTRY\_NODEF10(170), ERROR\_ENTRY\_NODEF10(180), ERROR\_ENTRY\_NODEF10(190),

ERROR\_ENTRY(200, "Лексическая ошибка: Недопустимый символ в исходном файле(-in)"),

ERROR\_ENTRY(201, "Лексическая ошибка: Неизвестная последовательность символов"),

ERROR\_ENTRY(202, "Лексическая ошибка: Превышен размер таблицы лексем"),

ERROR\_ENTRY(203, "Лексическая ошибка: Превышен размер таблицы идентификаторов"),

ERROR\_ENTRY(204, "Лексическая ошибка: Неверная запись шеснадцатеричного числа"),

ERROR\_ENTRY\_NODEF(205), ERROR\_ENTRY\_NODEF(206), ERROR\_ENTRY\_NODEF(207), ERROR\_ENTRY\_NODEF(208), ERROR\_ENTRY\_NODEF(209),

ERROR\_ENTRY\_NODEF10(210), ERROR\_ENTRY\_NODEF10(220), ERROR\_ENTRY\_NODEF10(230), ERROR\_ENTRY\_NODEF10(240),

ERROR\_ENTRY\_NODEF10(250), ERROR\_ENTRY\_NODEF10(260), ERROR\_ENTRY\_NODEF10(270), ERROR\_ENTRY\_NODEF10(280), ERROR\_ENTRY\_NODEF10(290),

ERROR\_ENTRY(300, "Семантическая ошибка: Необъявленный идентификатор"),

ERROR\_ENTRY(301, "Семантическая ошибка: Отсутствует точка входа main"),

ERROR\_ENTRY(302, "Семантическая ошибка: Обнаружено несколько точек входа main"),

ERROR\_ENTRY(303, "Семантическая ошибка: В объявлении не указан тип идентификатора"),

ERROR\_ENTRY(304, "Семантическая ошибка: В объявлении отсутствует ключевое слово new"),

ERROR\_ENTRY(305, "Семантическая ошибка: Попытка переопределения идентификатора"),

ERROR\_ENTRY(306, "Семантическая ошибка: Превышено максимальное количество параметров функции"),

ERROR\_ENTRY(307, "Семантическая ошибка: Слишком много параметров в вызове"),

ERROR\_ENTRY(308, "Семантическая ошибка: Кол-во ожидаемыех функцией и передаваемых параметров не совпадают"),

ERROR\_ENTRY(309, "Семантическая ошибка: Несовпадение типов передаваемых параметров"),

ERROR\_ENTRY(310, "Семантическая ошибка: Использование пустого строкового литерала недопустимо"),

ERROR\_ENTRY(311, "Семантическая ошибка: Обнаружен символ \'\"\'. Возможно, не закрыт строковый литерал"),

ERROR\_ENTRY(312, "Семантическая ошибка: Превышен размер строкового литерала"),

ERROR\_ENTRY(313, "Семантическая ошибка: Недопустимый целочисленный литерал"),

ERROR\_ENTRY(314, "Семантическая ошибка: Типы данных в выражении не совпадают"),

ERROR\_ENTRY(315, "Семантическая ошибка: Тип функции и возвращаемого значения не совпадают"),

ERROR\_ENTRY(316, "Семантическая ошибка: Недопустимое строковое выражение справа от знака \'=\'"),

ERROR\_ENTRY(317, "Семантическая ошибка: Неверное условное выражение"),

ERROR\_ENTRY(318, "Семантическая ошибка: Деление на ноль"),

ERROR\_ENTRY\_NODEF(319),

ERROR\_ENTRY\_NODEF10(320),ERROR\_ENTRY\_NODEF10(330),ERROR\_ENTRY\_NODEF10(340),ERROR\_ENTRY\_NODEF10(350),

ERROR\_ENTRY\_NODEF10(360),ERROR\_ENTRY\_NODEF10(370),ERROR\_ENTRY\_NODEF10(380),ERROR\_ENTRY\_NODEF10(390),

ERROR\_ENTRY\_NODEF100(400), ERROR\_ENTRY\_NODEF100(500),

ERROR\_ENTRY(600, "Синтаксическая ошибка: Неверная структура программы"),

ERROR\_ENTRY(601, "Синтаксическая ошибка: Не найден список параметров функции"),

ERROR\_ENTRY(602, "Синтаксическая ошибка: Ошибка в теле функции"),

ERROR\_ENTRY(603, "Синтаксическая ошибка: Ошибка в теле скрипта"),

ERROR\_ENTRY(604, "Синтаксическая ошибка: Ошибка в списке параметров функции"),

ERROR\_ENTRY(605, "Синтаксическая ошибка: Ошибка в вызове функции/выражении"),

ERROR\_ENTRY(606, "Синтаксическая ошибка: Ошибка в списке фактических параметров функции"),

ERROR\_ENTRY(607, "Синтаксическая ошибка: Ошибка при констуировании цикла/условного выражения"),

ERROR\_ENTRY(608, "Синтаксическая ошибка: Ошибка в теле цикла/условного выражения"),

ERROR\_ENTRY(609, "Синтаксическая ошибка: Ошибка в условии цикла/условного выражения"),

ERROR\_ENTRY(610, "Синтаксическая ошибка: Неверный условный оператор"),

ERROR\_ENTRY(611, "Синтаксическая ошибка: Неверный арифметический оператор"),

ERROR\_ENTRY(612, "Синтаксическая ошибка: Неверное выражение. Ожидаются только идентификаторы/литералы"),

ERROR\_ENTRY(613, "Синтаксическая ошибка: Ошибка в арифметическом выражении"),

ERROR\_ENTRY(614, "Синтаксическая ошибка: Недопустимая синтаксическая конструкция"),

ERROR\_ENTRY(615, "Синтаксическая ошибка: Недопустимая синтаксическая конструкция в теле цикла/условного выражения"),

Рисунок 1 Таблица ошибок языка NEV-2023

# Приложение Б

Листинг 1 Таблица идентификаторов контрольного примера

| N |СТРОКА В ТЛ | ТИП ИДЕНТИФИКАТОРА| ИМЯ | ЗНАЧЕНИЕ (ПАРАМЕТРЫ)

| 0 | 2 | number function | min | P0:NUMBER | P1:NUMBER |

| 1 | 5 | number parameter | minx |

| 2 | 8 | number parameter | miny |

| 3 | 13 | number variable | minres |0

| 4 | 36 | string literal | LTRL1 |[21]Minimal number is:

| 5 | 51 | number function | div | P0:NUMBER |

| 6 | 54 | number parameter | divnum |

| 7 | 61 | number literal | LTRL2 |3

| 8 | 67 | string literal | LTRL3 |[2];

| 9 | 75 | void function | standf | P0:STRING | P1:STRING | P2:STRING |

| 10 | 78 | string parameter | standfa |

| 11 | 81 | string parameter | standfb |

| 12 | 84 | string parameter | standfc |

| 13 | 88 | string literal | LTRL4 |[21]Standart function:

| 14 | 94 | number variable | standfk |0

| 15 | 98 | number LIB FUNC | lenght | P0:STRING |

| 16 | 104 | string literal | LTRL5 |[21]Lenght NEV-2023:

| 17 | 113 | string variable | standfstr |[0]

| 18 | 117 | string LIB FUNC | strcopy | P0:STRING |

| 19 | 123 | string literal | LTRL6 |[21]String copy:

| 20 | 132 | number variable | standfnum |0

| 21 | 136 | number LIB FUNC | atoii | P0:STRING |

| 22 | 142 | string literal | LTRL7 |[21]String in number:

| 23 | 158 | number variable | mainx |0

| 24 | 160 | number literal | LTRL8 |10

| 25 | 164 | number variable | mainy |0

| 26 | 175 | string literal | LTRL9 |[21]10 module 3:

| 27 | 186 | string variable | mainstrx |[0]

| 28 | 188 | string literal | LTRL10 |[8]NEV-2023

| 29 | 192 | string variable | mainstry |[0]

| 30 | 194 | string literal | LTRL11 |[19]Course Project Test

| 31 | 198 | string variable | mainstrz |[0]

| 32 | 200 | string literal | LTRL12 |[2]70

| 33 | 213 | number variable | mainminone|0

| 34 | 215 | number literal | LTRL13 |21

| 35 | 219 | number variable | mainmintwo|0

| 36 | 221 | number literal | LTRL14 |13

| 37 | 225 | number variable | mainresmin|0

| 38 | 237 | string literal | LTRL15 |[21]Cycle:

| 39 | 243 | number variable | mainab |0

| 40 | 245 | number literal | LTRL16 |125

| 41 | 248 | string literal | LTRL17 |[21]Number before cycle:

| 42 | 258 | number literal | LTRL18 |1

| 43 | 274 | string literal | LTRL19 |[21]Number after cycle:

Листинг 2 Таблица лексем после контрольного примера

| 48 | } | 13 | |

| 49 | t | 15 | |

| 50 | f | 15 | |

| 51 | i | 15 | 5 |

| 52 | ( | 15 | |

| 53 | t | 15 | |

| 54 | i | 15 | 6 |

| 55 | ) | 15 | |

| 56 | { | 16 | |

| 57 | i | 17 | 6 |

| 58 | = | 17 | |

| 59 | i | 17 | 6 |

| 60 | / | 17 | |

| 61 | l | 17 | 7 |

| 62 | ; | 17 | |

| 63 | o | 18 | |

| 64 | i | 18 | 6 |

| 65 | ; | 18 | |

| 66 | o | 19 | |

| 67 | l | 19 | 8 |

| 68 | ; | 19 | |

| 69 | e | 20 | |

| 70 | i | 20 | 6 |

| 71 | ; | 20 | |

| 72 | } | 21 | |

| 73 | p | 23 | |

| 74 | f | 23 | |

| 75 | i | 23 | 9 |

| 76 | ( | 23 | |

| 77 | t | 23 | |

| 78 | i | 23 | 10 |

| 79 | , | 23 | |

| 80 | t | 23 | |

| 81 | i | 23 | 11 |

| 82 | , | 23 | |

| 83 | t | 23 | |

| 84 | i | 23 | 12 |

| 85 | ) | 23 | |

| 86 | { | 24 | |

| 87 | o | 25 | |

| 88 | l | 25 | 13 |

| 89 | ; | 25 | |

| 90 | ^ | 26 | |

| 91 | ; | 26 | |

| 92 | n | 28 | |

| 93 | t | 28 | |

| 94 | i | 28 | 14 |

| 95 | ; | 28 | |

| 96 | i | 29 | 14 |

| 97 | = | 29 | |

| N | ЛЕКСЕМА | СТРОКА | ИНДЕКС В ТИ |

| 0 | t | 1 | |

| 1 | f | 1 | |

| 2 | i | 1 | 0 |

| 3 | ( | 1 | |

| 4 | t | 1 | |

| 5 | i | 1 | 1 |

| 6 | , | 1 | |

| 7 | t | 1 | |

| 8 | i | 1 | 2 |

| 9 | ) | 1 | |

| 10 | { | 2 | |

| 11 | n | 3 | |

| 12 | t | 3 | |

| 13 | i | 3 | 3 |

| 14 | ; | 3 | |

| 15 | ? | 4 | |

| 16 | i | 4 | 1 |

| 17 | < | 4 | |

| 18 | i | 4 | 2 |

| 19 | # | 5 | |

| 20 | w | 6 | |

| 21 | { | 6 | |

| 22 | i | 6 | 3 |

| 23 | = | 6 | |

| 24 | i | 6 | 1 |

| 25 | ; | 6 | |

| 26 | } | 6 | |

| 27 | r | 7 | |

| 28 | { | 7 | |

| 29 | i | 7 | 3 |

| 30 | = | 7 | |

| 31 | i | 7 | 2 |

| 32 | ; | 7 | |

| 33 | } | 7 | |

| 34 | # | 8 | |

| 35 | o | 9 | |

| 36 | l | 9 | 4 |

| 37 | ; | 9 | |

| 38 | o | 10 | |

| 39 | i | 10 | 3 |

| 40 | ; | 10 | |

| 41 | ^ | 11 | |

| 42 | ; | 11 | |

| 43 | ^ | 11 | |

| 44 | ; | 11 | |

| 45 | e | 12 | |

| 46 | i | 12 | 3 |

| 47 | ; | 12 | |

Листинг 2 (продолжение) Таблица лексем после контрольного примера

| 98 | i | 29 | 15 |

| 99 | ( | 29 | |

|100 | i | 29 | 10 |

|101 | ) | 29 | |

|102 | ; | 29 | |

|103 | o | 30 | |

|104 | l | 30 | 16 |

|105 | ; | 30 | |

|106 | o | 31 | |

|107 | i | 31 | 14 |

|108 | ; | 31 | |

|109 | ^ | 32 | |

|110 | ; | 32 | |

|111 | n | 34 | |

|112 | t | 34 | |

|113 | i | 34 | 17 |

|114 | ; | 34 | |

|115 | i | 35 | 17 |

|116 | = | 35 | |

|117 | i | 35 | 18 |

|118 | ( | 35 | |

|119 | i | 35 | 11 |

|120 | ) | 35 | |

|121 | ; | 35 | |

|122 | o | 36 | |

|123 | l | 36 | 19 |

|124 | ; | 36 | |

|125 | o | 37 | |

|126 | i | 37 | 17 |

|127 | ; | 37 | |

|128 | ^ | 38 | |

|129 | ; | 38 | |

|130 | n | 40 | |

|131 | t | 40 | |

|132 | i | 40 | 20 |

|133 | ; | 40 | |

|134 | i | 41 | 20 |

|135 | = | 41 | |

|136 | i | 41 | 21 |

|137 | ( | 41 | |

|138 | i | 41 | 12 |

139 | ) | 41 | |

|140 | ; | 41 | |

|141 | o | 42 | |

|142 | l | 42 | 22 |

|143 | ; | 42 | |

|144 | o | 43 | |

|145 | i | 43 | 20 |

|146 | ; | 43 | |

|147 | ^ | 44 | |

|148 | ; | 44 | |

|149 | ^ | 44 | |

|150 | ; | 44 | |

|151 | e | 45 | |

|152 | ; | 45 | |

|153 | } | 46 | |

|154 | m | 48 | |

|155 | { | 49 | |

|156 | n | 50 | |

|157 | t | 50 | |

|158 | i | 50 | 23 |

|159 | = | 50 | |

|160 | l | 50 | 24 |

|161 | ; | 50 | |

|162 | n | 51 | |

|163 | t | 51 | |

|164 | i | 51 | 25 |

|165 | = | 51 | |

|166 | l | 51 | 7 |

|167 | ; | 51 | |

|168 | i | 52 | 23 |

|169 | = | 52 | |

|170 | i | 52 | 23 |

|171 | % | 52 | |

|172 | i | 52 | 25 |

|173 | ; | 52 | |

|174 | o | 53 | |

|175 | l | 53 | 26 |

|176 | ; | 53 | |

|177 | o | 54 | |

|178 | i | 54 | 23 |

|179 | ; | 54 | |

|180 | ^ | 55 | |

|181 | ; | 55 | |

|182 | ^ | 55 | |

|183 | ; | 55 | |

|184 | n | 57 | |

|185 | t | 57 | |

|186 | i | 57 | 27 |

|187 | = | 57 | |

|188 | l | 57 | 28 |

|189 | ; | 57 | |

|190 | n | 58 | |

|191 | t | 58 | |

|192 | i | 58 | 29 |

|193 | = | 58 | |

|194 | l | 58 | 30 |

|195 | ; | 58 | |

|196 | n | 59 | |

|197 | t | 59 | |

|198 | i | 59 | 31 |

|199 | = | 59 | |

Листинг 2 (продолжение) Таблица лексем после контрольного примера

|248 | l | 71 | 41 |

|249 | ; | 71 | |

|250 | o | 72 | |

|251 | i | 72 | 39 |

|252 | ; | 72 | |

|253 | ^ | 73 | |

|254 | ; | 73 | |

|255 | ? | 74 | |

|256 | i | 74 | 39 |

|257 | > | 74 | |

|258 | l | 74 | 42 |

|259 | # | 75 | |

|260 | c | 76 | |

|261 | { | 77 | |

|262 | i | 78 | 39 |

|263 | = | 78 | |

|264 | i | 78 | 5 |

|265 | ( | 78 | |

|266 | i | 78 | 39 |

|267 | ) | 78 | |

|268 | ; | 78 | |

|269 | } | 79 | |

|270 | # | 80 | |

|271 | ^ | 81 | |

|272 | ; | 81 | |

|273 | o | 82 | |

|274 | l | 82 | 43 |

|275 | ; | 82 | |

|276 | o | 83 | |

|277 | i | 83 | 39 |

|278 | ; | 83 | |

|279 | ^ | 84 | |

|280 | ; | 84 | |

|281 | } | 87 | |

|200 | l | 59 | 32 |

|201 | ; | 59 | |

|202 | i | 61 | 9 |

|203 | ( | 61 | |

|204 | i | 61 | 27 |

|205 | , | 61 | |

|206 | i | 61 | 29 |

|207 | , | 61 | |

|208 | i | 61 | 31 |

|209 | ) | 61 | |

|210 | ; | 61 | |

|211 | n | 63 | |

|212 | t | 63 | |

|213 | i | 63 | 33 |

|214 | = | 63 | |

|215 | l | 63 | 34 |

|216 | ; | 63 | |

|217 | n | 64 | |

|218 | t | 64 | |

|219 | i | 64 | 35 |

|220 | = | 64 | |

|221 | l | 64 | 36 |

|222 | ; | 64 | |

|223 | n | 65 | |

|224 | t | 65 | |

|225 | i | 65 | 37 |

|226 | ; | 65 | |

|227 | i | 66 | 37 |

|228 | = | 66 | |

|229 | i | 66 | 0 |

|230 | ( | 66 | |

|231 | i | 66 | 33 |

|232 | , | 66 | |

|233 | i | 66 | 35 |

|234 | ) | 66 | |

|235 | ; | 66 | |

|236 | o | 68 | |

|237 | l | 68 | 38 |

|238 | ; | 68 | |

|239 | ^ | 69 | |

|240 | ; | 69 | |

|241 | n | 70 | |

|242 | t | 70 | |

|243 | i | 70 | 39 |

|244 | = | 70 | |

|245 | l | 70 | 40 |

|246 | ; | 70 | |

|247 | o | 71 | |

# Приложение В

Листинг 1 Грамматика языка NEV-2023

Greibach greibach(NS('S'), TS('$'), 16,

Rule(NS('S'), GRB\_ERROR\_SERIES, 3,

Rule::Chain(6, TS('t'), TS('f'), TS('i'), NS('P'), NS('T'), NS('S')),

Rule::Chain(6, TS('p'), TS('f'), TS('i'), NS('P'), NS('G'), NS('S')),

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

),

Rule(NS('T'), GRB\_ERROR\_SERIES + 2, 2,

Rule::Chain(5, TS('{'), TS('e'), NS('V'), TS(';'), TS('}')),

Rule::Chain(6, TS('{'), NS('K'), TS('e'), NS('V'), TS(';'), TS('}'))

),

Rule(NS('G'), GRB\_ERROR\_SERIES + 3, 2,

Rule::Chain(4, TS('{'), TS('e'), TS(';'), TS('}')),

Rule::Chain(5, TS('{'), NS('K'), TS('e'), TS(';'), TS('}'))

),

Rule(NS('P'), GRB\_ERROR\_SERIES + 1, 2,

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

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

),

Rule(NS('E'), GRB\_ERROR\_SERIES + 4, 2,

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

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

),

Rule(NS('F'), GRB\_ERROR\_SERIES + 5, 2,

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

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

),

Rule(NS('N'), GRB\_ERROR\_SERIES + 6, 4,

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

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

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

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

),

Rule(NS('R'), GRB\_ERROR\_SERIES + 7, 5,

Rule::Chain(3, TS('r'), NS('Y'), TS('#')),

Rule::Chain(3, TS('w'), NS('Y'), TS('#')),

Rule::Chain(3, TS('c'), NS('Y'), TS('#')),

Rule::Chain(5, TS('r'), NS('Y'), TS('w'), NS('Y'), TS('#')),

Rule::Chain(5, TS('w'), NS('Y'), TS('r'), NS('Y'), TS('#'))

),

Rule(NS('Y'), GRB\_ERROR\_SERIES + 8, 1,

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

),

Rule(NS('Z'), GRB\_ERROR\_SERIES + 9, 3,

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

Rule::Chain(3, TS('i'), NS('L'), TS('l')),

Rule::Chain(3, TS('l'), NS('L'), TS('i'))

),

Rule(NS('L'), GRB\_ERROR\_SERIES + 10, 4,

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

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

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

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

),

Листинг 1(продолжение) Грамматика языка NEV-2023

Rule(NS('A'), GRB\_ERROR\_SERIES + 11, 5,

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

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

Rule::Chain(1, TS('\*')),

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

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

),

Rule(NS('V'), GRB\_ERROR\_SERIES + 12, 3,

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

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

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

),

Rule(NS('W'), GRB\_ERROR\_SERIES + 13, 8,

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

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

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

Rule::Chain(5, TS('('), NS('W'), TS(')'), NS('A'), NS('W')),

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

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

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

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

),

Rule(NS('K'), GRB\_ERROR\_SERIES + 14, 14,

Rule::Chain(7, TS('n'), TS('t'), TS('i'), TS('='), NS('V'), TS(';'), NS('K')),

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

Rule::Chain(5, TS('i'), TS('='), NS('W'), TS(';'), NS('K')),

Rule::Chain(4, TS('o'), NS('V'), TS(';'), NS('K')),

Rule::Chain(3, TS('^'), TS(';'), NS('K')),

Rule::Chain(5, TS('?'), NS('Z'), TS('#'), NS('R'), NS('K')),

Rule::Chain(4, TS('i'), NS('F'), TS(';'), NS('K')),

Rule::Chain(6, TS('n'), TS('t'), TS('i'), TS('='), NS('V'), TS(';')),

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

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

Rule::Chain(3, TS('o'), NS('V'), TS(';')),

Rule::Chain(2, TS('^'), TS(';')),

Rule::Chain(4, TS('?'), NS('Z'), TS('#'), NS('R')),

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

),

Rule(NS('X'), GRB\_ERROR\_SERIES + 15, 8,

Rule::Chain(5, TS('i'), TS('='), NS('W'), TS(';'), NS('X')),

Rule::Chain(4, TS('o'), NS('V'), TS(';'), NS('X')),

Rule::Chain(3, TS('^'), TS(';'), NS('X')),

Rule::Chain(4, TS('i'), NS('F'), TS(';'), NS('X')),

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

Rule::Chain(3, TS('o'), NS('V'), TS(';')),

Rule::Chain(2, TS('^'), TS(';')),

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

)

);

Листинг 2 Структура магазинного автомата

struct MfstState

{

short lenta\_position;

short nrule;

short nrulechain;

MFSTSTSTACK st;

MfstState();

MfstState(

short pposition,

MFSTSTSTACK pst,

short pnrulechain

);

MfstState(

short pposition,

MFSTSTSTACK 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 prc\_step,

short pnrule,

short pnrule\_chain

);

}

diagnosis[MFST\_DIAGN\_NUMBER];

GRBALPHABET\* lenta;

short lenta\_position;

short nrule;

short nrulechain;

short lenta\_size;

GRB::Greibach grebach;

Lexer::LEX lex;

MFSTSTSTACK st;

std::stack<MfstState> storestate;

Mfst();

Mfst(

Lexer::LEX 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(const Log::LOG& log);

Листинг 2 (продолжение) Структура магазинного автомата

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

bool push\_chain(

GRB::Rule::Chain chain

);

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

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

bool savediagnois(

RC\_STEP pprc\_step

);

void printrules(const Log::LOG& log);

struct Deducation

{

short size;

short\* nrules;

short\* nrulechains;

Deducation() { size = 0; nrules = 0; nrulechains = 0; };

} deducation;

bool savededucation();

};

Листинг 3 Структура грамматики Грейбах

struct Greibach

{

short size;

GRBALPHABET startN;

GRBALPHABET stbottomT;

Rule\* rules;

Greibach() { short size = 0; startN = 0; stbottomT = 0; rules = 0; };

Greibach(

GRBALPHABET pstartN,

GRBALPHABET pstbootomT,

short psize,

Rule r, ...

);

short getRule(

GRBALPHABET pnn,

Rule& prule

);

Rule getRule(short n);

};

Greibach getGreibach();

Листинг 4 Разбор исходного кода синтаксическим анализатором

Шаг :Правило Входная лента Стек

0 : S->tfiPTS tfi(ti,ti){nti;?i<i#w{i=i S$

1 : SAVESTATE: 1

1 : tfi(ti,ti){nti;?i<i#w{i=i tfiPTS$

2 : fi(ti,ti){nti;?i<i#w{i=i; fiPTS$

3 : i(ti,ti){nti;?i<i#w{i=i;} iPTS$

4 : (ti,ti){nti;?i<i#w{i=i;}r PTS$

5 : P->(E) (ti,ti){nti;?i<i#w{i=i;}r PTS$

6 : SAVESTATE: 2

6 : (ti,ti){nti;?i<i#w{i=i;}r (E)TS$

7 : ti,ti){nti;?i<i#w{i=i;}r{ E)TS$

8 : E->ti,E ti,ti){nti;?i<i#w{i=i;}r{ E)TS$

9 : SAVESTATE: 3

9 : ti,ti){nti;?i<i#w{i=i;}r{ ti,E)TS$

10 : i,ti){nti;?i<i#w{i=i;}r{i i,E)TS$

11 : ,ti){nti;?i<i#w{i=i;}r{i= ,E)TS$

12 : ti){nti;?i<i#w{i=i;}r{i=i E)TS$

13 : E->ti,E ti){nti;?i<i#w{i=i;}r{i=i E)TS$

14 : SAVESTATE: 4

14 : ti){nti;?i<i#w{i=i;}r{i=i ti,E)TS$

15 : i){nti;?i<i#w{i=i;}r{i=i; i,E)TS$

16 : ){nti;?i<i#w{i=i;}r{i=i;} ,E)TS$

17 : TS\_NOK/NS\_NORULECHАIN

17 : RESSTATE

17 : ti){nti;?i<i#w{i=i;}r{i=i E)TS$

18 : E->ti ti){nti;?i<i#w{i=i;}r{i=i E)TS$

19 : SAVESTATE: 4

19 : ti){nti;?i<i#w{i=i;}r{i=i ti)TS$

20 : i){nti;?i<i#w{i=i;}r{i=i; i)TS$

21 : ){nti;?i<i#w{i=i;}r{i=i;} )TS$

22 : {nti;?i<i#w{i=i;}r{i=i;}# TS$

23 : T->{eV;} {nti;?i<i#w{i=i;}r{i=i;}# TS$

24 : SAVESTATE: 5

24 : {nti;?i<i#w{i=i;}r{i=i;}# {eV;}S$

25 : nti;?i<i#w{i=i;}r{i=i;}#o eV;}S$

26 : TS\_NOK/NS\_NORULECHАIN

26 : RESSTATE

Листинг 4 (продолжение) Разбор исходного кода синтаксическим анализатором

843 : SAVESTATE: 132

843 : ^;ol;oi;^;} ^;K}$

844 : ;ol;oi;^;} ;K}$

845 : ol;oi;^;} K}$

846 : K->oV;K ol;oi;^;} K}$

847 : SAVESTATE: 133

847 : ol;oi;^;} oV;K}$

848 : l;oi;^;} V;K}$

849 : V->l l;oi;^;} V;K}$

850 : SAVESTATE: 134

850 : l;oi;^;} l;K}$

851 : ;oi;^;} ;K}$

852 : oi;^;} K}$

853 : K->oV;K oi;^;} K}$

854 : SAVESTATE: 135

854 : oi;^;} oV;K}$

855 : i;^;} V;K}$

856 : V->i i;^;} V;K}$

857 : SAVESTATE: 136

857 : i;^;} i;K}$

858 : ;^;} ;K}$

859 : ^;} K}$

860 : K->^;K ^;} K}$

861 : SAVESTATE: 137

861 : ^;} ^;K}$

862 : ;} ;K}$

863 : } K}$

864 : TNS\_NORULECHAIN/NS\_NORULE

864 : RESSTATE

864 : ^;} K}$

865 : K->^; ^;} K}$

866 : SAVESTATE: 137

866 : ^;} ^;}$

867 : ;} ;}$

868 : } }$

869 : $

870 : LENTA\_END

871 : ------>LENTA\_END

# Приложение Г

Листинг 1 Программная реализация механизма преобразования в ПОЛИЗ

bool \_\_cdecl setPolishNotation(IT::IdTable& idtable, Log::LOG& log, int lextable\_pos, ltvec& v)

{

vector < LT::Entry > result;

stack < LT::Entry > s;

bool ignore = false;

for (unsigned i = 0; i < v.size(); i++)

{

if (ignore)

{

result.push\_back(v[i]);

if (v[i].lexema == LEX\_RIGHTTHESIS)

ignore = false;

continue;

}

int priority = getPriority(v[i]);

if (v[i].lexema == LEX\_LEFTHESIS || v[i].lexema == LEX\_RIGHTTHESIS || v[i].lexema == LEX\_PLUS || v[i].lexema == LEX\_MINUS || v[i].lexema

== LEX\_STAR || v[i].lexema == LEX\_DIRSLASH || v[i].lexema == LEX\_MOD)

{

if (s.empty() || v[i].lexema == LEX\_LEFTHESIS)

{

s.push(v[i]);

continue;

}

if (v[i].lexema == LEX\_RIGHTTHESIS)

{

while (!s.empty() && s.top().lexema != LEX\_LEFTHESIS)

{

result.push\_back(s.top());

s.pop();

}

if (!s.empty() && s.top().lexema == LEX\_LEFTHESIS)

s.pop();

continue;

}

while (!s.empty() && getPriority(s.top()) >= priority)

{

result.push\_back(s.top());

s.pop();

}

s.push(v[i]);

}

if (v[i].lexema == LEX\_LITERAL || v[i].lexema == LEX\_ID)

{

if (idtable.table[v[i].idxTI].idtype == IT::IDTYPE::F || idtable.table[v[i].idxTI].idtype == IT::IDTYPE::S)

ignore = true;

result.push\_back(v[i]);

}

if (v[i].lexema != LEX\_LEFTHESIS && v[i].lexema != LEX\_RIGHTTHESIS && v[i].lexema != LEX\_PLUS && v[i].lexema

!= LEX\_MINUS && v[i].lexema != LEX\_STAR && v[i].lexema != LEX\_DIRSLASH && v[i].lexema != LEX\_ID && v[i].lexema

!= LEX\_LITERAL && v[i].lexema != LEX\_MOD)

{

Log::writeError(log.stream, Error::GetError(1));

return false;

}

}

while (!s.empty()) { result.push\_back(s.top()); s.pop(); }

v = result;

return true;

}

Листинг 2 Таблица идентификаторов после преобразования выражений в ПОЛИЗ

N |СТРОКА В ТЛ| ТИП ИДЕНТИФИКАТОРА | ИМЯ | ЗНАЧЕНИЕ (ПАРАМЕТРЫ)

|0 | 2 | number function | min | P0:NUMBER | P1:NUMBER |

|1 | 5 | number parameter | minx |

|2 | 8 | number parameter | miny |

|3 | 13 | number variable | minres |0

|4 | 36 | string literal | LTRL1 |[21]Minimal number is:

|5 | 51 | number function | div | P0:NUMBER |

|6 | 54 | number parameter | divnum |

|7 | 60 | number literal | LTRL2 |3

|8 | 67 | string literal | LTRL3 |[2];

|9 | 75 | void function | standf | P0:STRING | P1:STRING | P2:STRING |

|10 | 78 | string parameter | standfa |

|11 | 81 | string parameter | standfb |

|12 | 84 | string parameter | standfc |

|13 | 88 | string literal | LTRL4 |[21]Standart function:

|14 | 94 | number variable | standfk |0

|15 | 98 | number LIB FUNC | lenght | P0:STRING |

|16 | 104 | string literal | LTRL5 |[21]Lenght NEV-2023:

|17 | 113 | string variable | standfstr |[0]

|18 | 117 | string LIB FUNC | strcopy | P0:STRING |

|19 | 123 | string literal | LTRL6 |[21]String copy:

|20 | 132 | number variable | standfnum |0

|21 | 136 | number LIB FUNC | atoii | P0:STRING |

|22 | 142 | string literal | LTRL7 |[21]String in number:

|23 | 158 | number variable | mainx |0

|24 | 160 | number literal | LTRL8 |10

|25 | 164 | number variable | mainy |0

|26 | 175 | string literal | LTRL9 |[21]10 module 3:

|27 | 186 | string variable | mainstrx |[0]

|28 | 188 | string literal | LTRL10 |[8]NEV-2023

|29 | 192 | string variable | mainstry |[0]

|30 | 194 | string literal | LTRL11 |[19]Course Project Test

|31 | 198 | string variable | mainstrz |[0]

|32 | 200 | string literal | LTRL12 |[2]70

|33 | 213 | number variable | mainminone |0

|34 | 215 | number literal | LTRL13 |21

|35 | 219 | number variable | mainmintwo |0

|36 | 221 | number literal | LTRL14 |13

|37 | 225 | number variable | mainresmin |0

|38 | 237 | string literal | LTRL15 |[21]Cycle:

|39 | 243 | number variable | mainab |0

|40 | 245 | number literal | LTRL16 |125

|41 | 248 | string literal | LTRL17 |[21]Number before cycle:

|42 | 258 | number literal | LTRL18 |1

|43 | 274 | string literal | LTRL19 |[21]Number after cycle:

Листинг 3 Таблица лексем после преобразования выражений в ПОЛИЗ

N | ЛЕКСЕМА | СТРОКА | ИНДЕКС В ТИ |

|0 | t | 1 | |

|1 | f | 1 | |

|2 | i | 1 |0 |

|3 | ( | 1 | |

|4 | t | 1 | |

|5 | i | 1 |1 |

|6 | , | 1 | |

|7 | t | 1 | |

|8 | i | 1 |2 |

|9 | ) | 1 | |

|10 | { | 2 | |

|11 | n | 3 | |

|12 | t | 3 | |

|13 | i | 3 |3 |

|14 | ; | 3 | |

|15 | ? | 4 | |

|16 | i | 4 |1 |

|17 | < | 4 | |

|18 | i | 4 |2 |

|19 | # | 5 | |

|20 | w | 6 | |

|21 | { | 6 | |

|22 | i | 6 |3 |

|23 | = | 6 | |

|24 | i | 6 |1 |

|25 | ; | 6 | |

|26 | } | 6 | |

|27 | r | 7 | |

|28 | { | 7 | |

|29 | i | 7 |3 |

|30 | = | 7 | |

|31 | i | 7 |2 |

|32 | ; | 7 | |

|33 | } | 7 | |

|34 | # | 8 | |

|35 | o | 9 | |

|36 | l | 9 |4 |

|37 | ; | 9 | |

|38 | o | 10 | |

|39 | i | 10 |3 |

|40 | ; | 10 | |

|41 | ^ | 11 | |

|42 | ; | 11 | |

|43 | ^ | 11 | |

|44 | ; | 11 | |

|45 | e | 12 | |

|46 | i | 12 |3 |

|47 | ; | 12 | |

|48 | } | 13 | |

|49 | t | 15 | |

|50 | f | 15 | |

|51 | i | 15 |5 |

|52 | ( | 15 | |

|53 | t | 15 | |

|54 | i | 15 |6 |

|55 | ) | 15 | |

|56 | { | 16 | |

|57 | i | 17 |6 |

|58 | = | 17 | |

|59 | i | 17 |6 |

|60 | l | 17 |7 |

|61 | / | 17 | |

|62 | ; | 17 | |

|63 | o | 18 | |

|64 | i | 18 |6 |

|65 | ; | 18 | |

|66 | o | 19 | |

|67 | l | 19 |8 |

|68 | ; | 19 | |

|69 | e | 20 | |

|70 | i | 20 |6 |

|71 | ; | 20 | |

|72 | } | 21 | |

|73 | p | 23 | |

|74 | f | 23 | |

|75 | i | 23 |9 |

|76 | ( | 23 | |

|77 | t | 23 | |

|78 | i | 23 |10 |

|79 | , | 23 | |

|80 | t | 23 | |

|81 | i | 23 |11 |

|82 | , | 23 | |

|83 | t | 23 | |

|84 | i | 23 |12 |

|85 | ) | 23 | |

|86 | { | 24 | |

|87 | o | 25 | |

|88 | l | 25 |13 |

|89 | ; | 25 | |

|90 | ^ | 26 | |

|91 | ; | 26 | |

|92 | n | 28 | |

|93 | t | 28 | |

|94 | i | 28 |14 |

|95 | ; | 28 | |

|96 | i | 29 |14 |

|97 | = | 29 | |

|98 | i | 29 |15 |

|99 | ( | 29 | |

|100 | i | 29 |10 |

|101 | ) | 29 | |

|102 | ; | 29 | |

|103 | o | 30 | |

|104 | l | 30 |16 |

|105 | ; | 30 | |

|106 | o | 31 | |

|107 | i | 31 |14 |

|108 | ; | 31 | |

|109 | ^ | 32 | |

|110 | ; | 32 | |

|111 | n | 34 | |

|112 | t | 34 | |

|113 | i | 34 |17 |

|114 | ; | 34 | |

|115 | i | 35 |17 |

|116 | = | 35 | |

|117 | i | 35 |18 |

|118 | ( | 35 | |

|119 | i | 35 |11 |

|120 | ) | 35 | |

|121 | ; | 35 | |

|122 | o | 36 | |

|123 | l | 36 |19 |

|124 | ; | 36 | |

|125 | o | 37 | |

|126 | i | 37 |17 |

|127 | ; | 37 | |

|128 | ^ | 38 | |

|129 | ; | 38 | |

|130 | n | 40 | |

|131 | t | 40 | |

|132 | i | 40 |20 |

|133 | ; | 40 | |

|134 | i | 41 |20 |

|135 | = | 41 | |

|136 | i | 41 |21 |

|137 | ( | 41 | |

|138 | i | 41 |12 |

|139 | ) | 41 | |

|140 | ; | 41 | |

|141 | o | 42 | |

|142 | l | 42 |22 |

|143 | ; | 42 | |

|144 | o | 43 | |

|145 | i | 43 |20 |

|146 | ; | 43 | |

|147 | ^ | 44 | |

|148 | ; | 44 | |

|149 | ^ | 44 | |

|150 | ; | 44 | |

|151 | e | 45 | |

|152 | ; | 45 | |

|153 | } | 46 | |

|154 | m | 48 | |

|155 | { | 49 | |

|156 | n | 50 | |

|157 | t | 50 | |

|158 | i | 50 |23 |

|159 | = | 50 | |

|160 | l | 50 |24 |

|161 | ; | 50 | |

|162 | n | 51 | |

|163 | t | 51 | |

|164 | i | 51 |25 |

|165 | = | 51 | |

|166 | l | 51 |7 |

|167 | ; | 51 | |

|168 | i | 52 |23 |

|169 | = | 52 | |

|170 | i | 52 |23 |

|171 | i | 52 |25 |

|172 | % | 52 | |

|173 | ; | 52 | |

|174 | o | 53 | |

|175 | l | 53 |26 |

|176 | ; | 53 | |

|177 | o | 54 | |

|178 | i | 54 |23 |

|179 | ; | 54 | |

|180 | ^ | 55 | |

|181 | ; | 55 | |

|182 | ^ | 55 | |

|183 | ; | 55 | |

|184 | n | 57 | |

|185 | t | 57 | |

|186 | i | 57 |27 |

|187 | = | 57 | |

|188 | l | 57 |28 |

|189 | ; | 57 | |

|190 | n | 58 | |

|191 | t | 58 | |

|192 | i | 58 |29 |

|193 | = | 58 | |

|194 | l | 58 |30 |

|195 | ; | 58 | |

|196 | n | 59 | |

|197 | t | 59 | |

|198 | i | 59 |31 |

|199 | = | 59 | |

|200 | l | 59 |32 |

|201 | ; | 59 | |

|202 | i | 61 |9 |

|203 | ( | 61 | |

|204 | i | 61 |27 |

|205 | , | 61 | |

|206 | i | 61 |29 |

|207 | , | 61 | |

|208 | i | 61 |31 |

|209 | ) | 61 | |

|210 | ; | 61 | |

|211 | n | 63 | |

|212 | t | 63 | |

|213 | i | 63 |33 |

|214 | = | 63 | |

|215 | l | 63 |34 |

|216 | ; | 63 | |

|217 | n | 64 | |

|218 | t | 64 | |

|219 | i | 64 |35 |

|220 | = | 64 | |

|221 | l | 64 |36 |

|222 | ; | 64 | |

|223 | n | 65 | |

|224 | t | 65 | |

|225 | i | 65 |37 |

|226 | ; | 65 | |

|227 | i | 66 |37 |

|228 | = | 66 | |

|229 | i | 66 |0 |

|230 | ( | 66 | |

|231 | i | 66 |33 |

|232 | , | 66 | |

|233 | i | 66 |35 |

|234 | ) | 66 | |

|235 | ; | 66 | |

|236 | o | 68 | |

|237 | l | 68 |38 |

|238 | ; | 68 | |

|239 | ^ | 69 | |

|240 | ; | 69 | |

|241 | n | 70 | |

|242 | t | 70 | |

|243 | i | 70 |39 |

|244 | = | 70 | |

|245 | l | 70 |40 |

|246 | ; | 70 | |

|247 | o | 71 | |

|248 | l | 71 |41 |

|249 | ; | 71 | |

|250 | o | 72 | |

|251 | i | 72 |39 |

|252 | ; | 72 | |

|253 | ^ | 73 | |

|254 | ; | 73 | |

|255 | ? | 74 | |

|256 | i | 74 |39 |

|257 | > | 74 | |

|258 | l | 74 |42 |

|259 | # | 75 | |

|260 | c | 76 | |

|261 | { | 77 | |

|262 | i | 78 |39 |

|263 | = | 78 | |

|264 | i | 78 |5 |

|265 | ( | 78 | |

|266 | i | 78 |39 |

|267 | ) | 78 | |

|268 | ; | 78 | |

|269 | } | 79 | |

|270 | # | 80 | |

|271 | ^ | 81 | |

|272 | ; | 81 | |

|273 | o | 82 | |

|274 | l | 82 |43 |

|275 | ; | 82 | |

|276 | o | 83 | |

|277 | i | 83 |39 |

|278 | ; | 83 | |

|279 | ^ | 84 | |

|280 | ; | 84 | |

|281 | } | 87 | |

|

Листинг 4 Соответствие лексем исходному коду программы

46 | }

48 | m

49 | {

50 | nti[23]=l[24];

51 | nti[25]=l[7];

52 | i[23]=i[23]i[25]%;

53 | ol[26];

54 | oi[23];

55 | ^;^;

57 | nti[27]=l[28];

58 | nti[29]=l[30];

59 | nti[31]=l[32];

61 | i[9](i[27],i[29],i[31]);

63 | nti[33]=l[34];

64 | nti[35]=l[36];

65 | nti[37];

66 | i[37]=i[0](i[33],i[35]);

68 | ol[38];

69 | ^;

70 | nti[39]=l[40];

71 | ol[41];

72 | oi[39];

73 | ^;

74 | ?i[39]>l[42]

75 | #

76 | c

77 | {

78 | i[39]=i[5](i[39]);

79 | }

80 | #

81 | ^;

82 | ol[43];

83 | oi[39];

84 | ^;

87 | }

1 | tfi[0](ti[1],ti[2])

2 | {

3 | nti[3];

4 | ?i[1]<i[2]

5 | #

6 | w{i[3]=i[1];}

7 | r{i[3]=i[2];}

8 | #

9 | ol[4];

10 | oi[3];

11 | ^;^;

12 | ei[3];

13 | }

15 | tfi[5](ti[6])

16 | {

17 | i[6]=i[6]l[7]/;

18 | oi[6];

19 | ol[8];

20 | ei[6];

21 | }

23 | pfi[9](ti[10],ti[11],ti[12])

24 | {

25 | ol[13];

26 | ^;

28 | nti[14];

29 | i[14]=i[15](i[10]);

30 | ol[16];

31 | oi[14];

32 | ^;

34 | nti[17];

35 | i[17]=i[18](i[11]);

36 | ol[19];

37 | oi[17];

38 | ^;

40 | nti[20];

41 | i[20]=i[21](i[12]);

42 | ol[22];

43 | oi[20];

44 | ^;^;

45 | e;

# Приложение Д

Листинг 1 Результат генерации кода контрольного примера в Ассемблере

.586

.model flat, stdcall

includelib libucrt.lib

includelib kernel32.lib

includelib "../Debug/GenLib.lib

ExitProcess PROTO:DWORD

.stack 4096

outnum PROTO : DWORD

outstr PROTO : DWORD

strcopy PROTO : DWORD, : DWORD

lenght PROTO : DWORD, : DWORD

atoii PROTO : DWORD, : DWORD

.const

newline byte 13, 10, 0

LTRL1 byte 'Minimal number is: ', 0

LTRL2 sdword 3

LTRL3 byte '; ', 0

LTRL4 byte 'Standart function: ', 0

LTRL5 byte 'Lenght NEV-2023: ', 0

LTRL6 byte 'String copy: ', 0

LTRL7 byte 'String in number: ', 0

LTRL8 sdword 10

LTRL9 byte '10 module 3: ', 0

LTRL10 byte 'NEV-2023', 0

LTRL11 byte 'Course Project Test', 0

LTRL12 byte '70', 0

LTRL13 sdword 21

LTRL14 sdword 13

LTRL15 byte 'Cycle: ', 0

LTRL16 sdword 125

LTRL17 byte 'Number before cycle: ', 0

LTRL18 sdword 1

LTRL19 byte 'Number after cycle: ', 0

.data

temp sdword ?

buffer byte 256 dup(0)

minres sdword 0

standfk sdword 0

standfstr dword ?

standfnum sdword 0

mainx sdword 0

mainy sdword 0

mainstrx dword ?

mainstry dword ?

mainstrz dword ?

mainminone sdword 0

mainmintwo sdword 0

mainresmin sdword 0

mainab sdword 0

.code

;------------- min --------------

min PROC,

minx : sdword, miny : sdword

; -------- save registers -------

push ebx

push edx

; -------------------------------

mov edx, minx

cmp edx, miny

jl right1

jg wrong1

right1:

push minx

pop ebx

mov minres, ebx

jmp next1

wrong1:

push miny

pop ebx

mov minres, ebx

next1:

push offset LTRL1

call outstr

push minres

call outnum

push offset newline

call outstr

push offset newline

call outstr

; ------ restore registers ------

pop edx

pop ebx

; -------------------------------

mov eax, minres

ret

min ENDP

;---------------

;------------- div --------------

div PROC,

divnum : sdword

; -------- save registers -------

push ebx

push edx

; -------------------------------

push divnum

push LTRL2

pop ebx

pop eax

cdq

idiv ebx

push eax

pop ebx

mov divnum, ebx

push divnum

call outnum

push offset LTRL3

call outstr

; ------ restore registers ------

pop edx

pop ebx

; -------------------------------

Листинг 1 (прод.) Результат генерации кода контрольного примера в Ассемблере

mov eax, divnum

ret

div ENDP

;---------------

;------------- standf --------------

standf PROC,

standfa : dword, standfb : dword, standfc : dword

; -------- save registers -------

push ebx

push edx

; -------------------------------

push offset LTRL4

call outstr

push offset newline

call outstr

push standfa

push offset buffer

call lenght

push eax

pop ebx

mov standfk, ebx

push offset LTRL5

call outstr

push standfk

call outnum

push offset newline

call outstr

push standfb

push offset buffer

call strcopy

mov standfstr, eax

push offset LTRL6

call outstr

push standfstr

call outstr

push offset newline

call outstr

push standfc

push offset buffer

call atoii

push eax

pop ebx

mov standfnum, ebx

push offset LTRL7

call outstr

push standfnum

call outnum

push offset newline

call outstr

push offset newline

call outstr

; ------ restore registers ------

pop edx

pop ebx

; -------------------------------

ret

standf ENDP

;---------------

;------------- MAIN --------------

main PROC

push LTRL8

pop ebx

mov mainx, ebx

push LTRL2

pop ebx

mov mainy, ebx

push mainx

push mainy

pop ebx

mov edx, 0

pop eax

idiv ebx

push edx

mov eax, edx

pop ebx

mov mainx, ebx

push offset LTRL9

call outstr

push mainx

call outnum

push offset newline

call outstr

push offset newline

call outstr

mov mainstrx, offset LTRL10

mov mainstry, offset LTRL11

mov mainstrz, offset LTRL12

push mainstrz

push mainstry

push mainstrx

call standf

push LTRL13

pop ebx

mov mainminone, ebx

Листинг 1 (прод.) Результат генерации кода контрольного примера в Ассемблере

jg cycle2

jmp cyclenext2

cycle2:

push mainab

call div

push eax

pop ebx

mov mainab, ebx

mov edx, mainab

cmp edx, LTRL18

jg cycle2

cyclenext2:

push offset newline

call outstr

push offset LTRL19

call outstr

push mainab

call outnum

push offset newline

call outstr

push 0

call ExitProcess

main ENDP

end main

push LTRL14

pop ebx

mov mainmintwo, ebx

push mainmintwo

push mainminone

call min

push eax

pop ebx

mov mainresmin, ebx

push offset LTRL15

call outstr

push offset newline

call outstr

push LTRL16

pop ebx

mov mainab, ebx

push offset LTRL17

call outstr

push mainab

call outnum

push offset newline

call outstr

mov edx, mainab

cmp edx, LTRL18