МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 6-05-0612-01 “Программная инженерия”

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора FVG – 2024»

Выполнил студент Федорович Вадим Геннадьевич

(Ф.И.О.)

Руководитель проекта acc. Гончар Егор Андреевич

(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 acc. Гончар Егор Андреевич

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

Нормоконтролер acc. Гончар Егор Андреевич

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

Курсовой проект защищен с оценкой

Минск 2024

Содержание

[Введение 5](#_Toc185554734)

[1 Спецификация языка программирования 6](#_Toc185554735)

[1.1 Характеристика языка программирования 6](#_Toc185554736)

[1.2 Определение алфавит языка программирования 6](#_Toc185554737)

[1.3 Применяемые сепараторы 7](#_Toc185554738)

[1.4 Применяемые кодировки 7](#_Toc185554739)

[1.5 Типы данных 9](#_Toc185554740)

[1.6 Преобразование типов данных 10](#_Toc185554741)

[1.7 Идентификаторы 10](#_Toc185554742)

[1.8 Литералы 10](#_Toc185554743)

[1.9 Область видимости идентификаторов 11](#_Toc185554744)

[1.10 Инициализация данных 11](#_Toc185554745)

[1.11 Инструкции языка 11](#_Toc185554746)

[1.13 Выражения и их вычисления 13](#_Toc185554747)

[1.14 Программные конструкции языка 13](#_Toc185554748)

[1.15 Область видимости 13](#_Toc185554749)

[1.16 Семантические проверки 13](#_Toc185554750)

[1.17 Распределение оперативной памяти на этапе выполнения 14](#_Toc185554751)

[1.18 Стандартная библиотека и её состав 15](#_Toc185554752)

[1.19 Ввод и вывод данных 15](#_Toc185554753)

[1.20 Точка входа 15](#_Toc185554754)

[1.21 Препроцессор 16](#_Toc185554755)

[1.22 Соглашения о вызовах 16](#_Toc185554756)

[1.23 Объектный код 16](#_Toc185554757)

[1.24 Классификация сообщений транслятора 16](#_Toc185554758)

[1.25 Контрольный пример 16](#_Toc185554759)

[2 Структура транслятора 17](#_Toc185554760)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 17](#_Toc185554761)

[2.2 Перечень входных параметров транслятора 18](#_Toc185554762)

[2.3 Перечень протоколов, формируемых транслятором и их содержимое 18](#_Toc185554763)

[3 Разработка лексического анализатора 20](#_Toc185554764)

[3.1 Структура лексического анализатора 20](#_Toc185554765)

[3.2 Контроль входных символов 20](#_Toc185554766)

[3.3 Удаление избыточных символов 21](#_Toc185554767)

[3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов 21](#_Toc185554768)

[3.5 Основные структуры данных 22](#_Toc185554769)

[3.6 Структура и перечень сообщений лексического анализатора 23](#_Toc185554770)

[3.7 Принцип обработки ошибок 23](#_Toc185554771)

[3.8 Параметры лексического анализатора 23](#_Toc185554772)

[3.9 Алгоритм лексического анализа 23](#_Toc185554773)

[3.10 Контрольный пример 24](#_Toc185554774)

[4 Разработка синтаксического анализатора 25](#_Toc185554775)

[4.1 Структура синтаксического анализатора 25](#_Toc185554776)

[4.2 Контекстно свободная грамматика, описывающая синтаксис языка 25](#_Toc185554777)

[4.3 Построение конечного магазинного автомата 27](#_Toc185554778)

[4.3 Основные структуры данных 28](#_Toc185554779)

[4.5 Описание алгоритма синтаксического разбора 28](#_Toc185554780)

[4.6 Структура и перечень сообщений синтаксического анализатора 29](#_Toc185554781)

[4.7 Параметры синтаксического анализатора и режимы его работы 29](#_Toc185554782)

[4.8 Принцип обработки ошибок 29](#_Toc185554783)

[5 Разработка семантического анализатора 30](#_Toc185554784)

[5.1 Структура семантического анализатора 30](#_Toc185554785)

[5.2 Функции семантического анализатора 30](#_Toc185554786)

[5.3 Структура и перечень сообщений семантического анализатора 30](#_Toc185554787)

[5.4 Принцип обработки ошибок 31](#_Toc185554788)

[5.5 Контрольный пример 31](#_Toc185554789)

[6 Преобразование выражений 32](#_Toc185554790)

[6.1 Выражения, допускаемые языком 32](#_Toc185554791)

[6.2 Польская запись 32](#_Toc185554792)

[6.3 Программная реализация обработки выражений 33](#_Toc185554793)

[6.4 Контрольный пример 33](#_Toc185554794)

[7 Генерация кода 34](#_Toc185554795)

[7.1 Структура генератора кода 34](#_Toc185554796)

[7.2 Представление типов данных в оперативной памяти 34](#_Toc185554797)

[7.3 Статическая библиотека 35](#_Toc185554798)

[7.4 Алгоритм работы генератора кода 36](#_Toc185554799)

[7.5 Контрольный пример 39](#_Toc185554800)

[8 Тестирование транслятора 40](#_Toc185554801)

[8.1 Тестирование фазы проверки на допустимость символов 40](#_Toc185554802)

[8.2 Тестирование лексического анализатора 40](#_Toc185554803)

[8.3 Тестирование синтаксического анализатора 41](#_Toc185554804)

[Заключение 42](#_Toc185554805)

[Список использованных источников 43](#_Toc185554806)

[ПРИЛОЖЕНИЕ А 44](#_Toc185554807)

[ПРИЛОЖЕНИЕ Б 45](#_Toc185554808)

[ПРИЛОЖЕНИЕ В 46](#_Toc185554809)

[ПРИЛОЖЕНИЕ Г 53](#_Toc185554810)

[ПРИЛОЖЕНИЕ Д 55](#_Toc185554811)

[ПРИЛОЖЕНИЕ Е 57](#_Toc185554812)

[ПРИЛОЖЕНИЕ З 60](#_Toc185554813)

[ПРИЛОЖЕНИЕ И 61](#_Toc185554814)

# **Введение**

Задачей данного курсового проекта является разработка транслятора для своего языка программирования: FVG-2024.

Написание транслятора будет осуществляться на языке C++, при этом код на языке FVG-2024 будет транслироваться в язык ассемблера.

Транслятор FVG-2024 состоит из следующих частей:

– семантический анализатор;

– синтаксический анализатор;

– логический анализатор;

– генератор исходного кода на языке ассемблера.

Исходя из цели курсового проекта, были определены следующие задачи:

– разбработка спецификации языка программирования;

– разбратка структуры транслятора;

– разработка лексического и семантического анализаторов;

– разработка синтаксического анализатора;

– преобразование выражений;

– генерация кода на язык ассемблера;

– тестирование транслятора.

Решения каждой из поставленных задач будут приведены в соответствующих главах курсового проекта.

# **1 Спецификация языка программирования**

## **Характеристика языка программирования**

Язык программирования FVG-2024 – это универсальный язык высокого уровня. Поддерживает 4 типа данных: символьный (char), целочисленный (short),

cтроковый (string), целочисленный положительный (uint). Он является процедурным, компилируемым, не объектно-ориентированным. Язык строго типизируемый, что говорит о невозможности преобразования типов, транслируемым языком программирования. Также в языке отсутствуют пользовательские типы данных

## **Определение алфавит языка программирования**

Совокупность символов, применяемых в определенном языке, определяется как алфавит этого языка. Структура алфавита языка FVG-2024 описана формально в виде набора правил, представленных в таблице 1.1.

Таблица 1.1 – Алфавит языка программирования FVG-2024

|  |
| --- |
| <строчная буква латинского алфавита>::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z |
| <прописная буква латинского алфавита>::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z |
| <цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| <арифметическая операция>::= +|-|\*|: |
| <символ- сепаратор>::= “”|,|(|)|{|}|;|[|]’|’ |
| <логическая операция>::= >|<|=|\|/|&|^ |

Исходный код FVG-2024 может содержать символы латинского алфавита, цифры десятичной системы счисления (от 0 до 9), символы арифметических операций и операций сравнения, а также символы-разделители.

## **Применяемые сепараторы**

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

Таблица 1.1 – Сепараторы

|  |  |  |
| --- | --- | --- |
| Сепаратор | Название | Область применения |
| ‘ ‘ | Пробел | Допускается везде, кроме идентификаторов и ключевых слов |
| ; | Точка с запятой | Разделение конструкций |
| {…} | Фигурные скобки | Заключение программного блока |
| […] | Квадратные кавычки | Блок кода |
| (…) | Круглые скобки | Приоритет операций, параметры функции |
| “…” | Двойные кавычки | Строковый литерал |
| ‘…’ | Одинарные кавычки | Допускается везде, кроме идентификаторов и ключевых слов |
| = | Знак «равно» | Присваивание значения |
| , | Запятая | Разделение параметров |
| /  \  +  -  \*  : | Знаки «косая черта», «обратная косая черта», «плюс», «астерикс», «тильда», | Выражения |
| &  ^  <  > | «амперсанд», «циркумфлекс», знаки «больше» и «меньше» | Выражения в операторе цикла |

## **Применяемые кодировки**

Для написания исходного кода на языке программирования FVG-2024 используется кодировка Windows-1251, которая широко применяется для работы с кириллическими текстами в системах, основанных на Windows. Эта кодировка включает полный набор символов русского алфaвита, a тaкже лaтинские буквы, цифры, знaки пунктуaции и дополнительные специальные символы.

Символы в Windows-1251 кодируются в одном бaйте, что позволяет предстaвить до 256 различных символов. Коды от 0 до 127 соответствуют стaндaртным символaм ASCII, используемым в большинстве языков прогрaммировaния. Коды от 128 до 255 включaют рaсширенный нaбор символов, включaющий кириллические буквы, псевдогрaфику и дополнительные знaки.

Визуaльное предстaвление кодировки Windows-1251 можно увидеть в тaблице символов, которaя связывает числовые коды символов с их грaфическим отображением. Нaпример:

1) Код 192 (C0 в шестнaдцaтеричной системе) соответствует букве "A" (кириллица),

2) Код 224 (E0) — букве "a".

Тaблица символов особенно полезнa при прогрaммировaнии нa низком уровне, где требуется рaботa с числовыми кодaми, или при рaзрaботке прогрaмм для обрaботки текстов нa кириллице.



Рисунок 1.1 – Таблица кодировки Windows-1251

Применение Windows-1251 имеет историческую основу. Оно стало распространённым благодаря популярности операционных систем Windows на территории стран СНГ. Несмотря на активное использование современных кодировок, таких как UTF-8, Windows-1251 по-прежнему востребована в случаях, где требуется минимизация размера текстовых данных или обеспечение обратной совместимости с устаревшими системами.

## **Типы данных**

В языке FVG-2024 реализованы четыре типа данных: символьный(char), строковый(string), целочисленный (short), целочисленный беззнаковый (uint). Описание типов данных, предусмотренных в данным языке представлено в таблице 1.2.

Таблица 1.2 – Типы данных языка FVG-2024

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| Целочисленный тип данных short | Фундаментальный тип данных. Используется для работы с целочисленными положительными значениями. В памяти занимает 2 байта.  При попытке инициализации значением больше максимального, инициализируется максимальным. При попытке инициализации значением меньше минимального, инициализируется значением, которое равно разнице между максимальным и исходным значением.  Максимальное значение: 32767. Минимальное значение: -32768.  Инициализация по умолчанию: значение 0. |
| Строковый тип данных string | Фундаментальный тип данных. Используется для работы с набором символов, каждый символ в памяти занимает 1 байт. Инициализация по умолчанию: символ конца строки “\0”. |
| Символьный тип данных char | Фундаментальный тип данных. Используется для работы с символом, который в памяти занимает 1 байт. Инициализация по умолчанию: символ конца строки “\0”. |
| Байтовый тип данных uint | Фундаментальный тип данных. Используется для работы с целочисленными положительными значениями.  Диапазон значений: от 0 до 4 294 967 295 (для 32-битного типа).  Объем памяти: 4 байта (32 бита).  Максимальное значение: 4 294 967 295.  Минимальное значение: 0.  Инициализация по умолчанию: значение 0. |

Пользовательские типы данных не поддерживаются.

## **Преобразование типов данных**

Преобразование типов данных не поддерживается, т.е. язык является строготипизированным. Но в стандартной библиотеке есть функции преобразования типа short в тип char и наоборот.

## **Идентификаторы**

В имени идентификатора допускаются только символы латинского алфавита и знак «\_». Максимальная длина имени идентификатора – 26 символов. Максимальная длина имени идентификатора функции – 26 символов. Имя идентификатора не может совпадать с именем функции, уже содержащаяся в стандартной библиотеке.

## **Литералы**

С помощью литералов осуществляется инициализация переменных. В языке существует три типа литералов. Краткое описание литералов языка FVG-2024 представлено в таблице 1.3.

Таблица 1.3 – Описание литералов

|  |  |  |  |
| --- | --- | --- | --- |
| Тип литерала | Регулярное выражение | Описание | Пример |
| Целочисленный литерал | -?[1-9]+[0-9]\* | Целочисленные неотрицательные литералы, по умолчанию инициализируются 0. Литералы могут быть только rvalue. | var short sum  sum = 15;  15 – целочисленный литерал. |
| Символьный литерал | [a-z|A-Z|А-Я|0-9|!-/] | Символ, заключённый в ‘’ (одинарные кавычки), по умолчанию инициализируются пустой строкой. Литералы могут быть только rvalue. | var char symbol;  symbol = ‘T’  T – символьный литерал. |
| Строковый литерал | [a-z|A-Z|А-Я|0-9|!-/]\* | Строковые литералы, максимальная длина строки 255 символов | var string str;  str = “God forbid 6” |

Литералы являются константами и при генерации кода объявляются один раз.

## **Область видимости идентификаторов**

Область видимости «сверху вниз» (по принципу С++). В языке FVG-2024 требуется обязательное объявление переменной перед её инициализацией и последующим использованием. Все переменные должны находиться внутри программного блока. Имеется возможность объявления одинаковых переменных в разных блоках, т. к. переменные, объявленные в одной функции, недоступны в другой. Объявление функций стандартной библиотеки можно производить в любом месте кода.

## **Инициализация данных**

При объявлении переменной не допускается инициализация. Описание способов инициализации переменных языка FVG-2024 представлено в таблице 1.4.

Таблица 1.4 – Способы инициализации переменных

|  |  |  |
| --- | --- | --- |
| Конструкция | Описание | Пример |
| var <тип данных> <идентификатор>; | Автоматическая инициализация: переменные типа short инициализируются нулём, переменные типа char – пустым символом, uint – нулем, string - nullptr. | var short sum;  var symbol chr;  var string str; |
| <идентификатор> = <значение>; | Присваивание переменной значения. | sum = 9;  Gonchar = ‘W’; |

Соответствие типов проверяется на синтаксическом анализе.

## **Инструкции языка**

Все возможные инструкции языка программирования FVG-2024 представлены в общем виде в таблице 1.5.

Таблица 1.5 – Инструкции языка программирования FVG-2024

|  |  |
| --- | --- |
| Инструкция | Запись на языке FVG-2024 |
| Объявление переменной | var <тип данных> <идентификатор>; |
| Объявление функции | var <тип данных> function <идентификатор> (<тип данных> <идентификатор>, …) {<блок кода>}; |
| Присваивание | <идентификатор> = <значение>/<идентификатор>; |
| Объявление внешней функции | make\_F <тип данных> function <идентификатор> (<тип данных> <идентификатор>, …); |
| Блок инструкций | main  {  …  } |

Таблица 1.5 – Продолжение

|  |  |
| --- | --- |
| Возврат из подпрограммы | return <идентификатор> / <литерал>; |
| Условная инструкция | while(<условие>)[<блок кода>]; |
| Вывод данных | write <идентификатор> / <литерал>; |

Инструкции (кроме функции входа в программу) требуют закрывающую «;».

* 1. **Операции языка**

Язык программирования FVG-2024 может выполнять операции, представленные в таблице 1.6. При сдвиге число сдвига (второй операнд) также ограничивается младшими битами. Для 16-битного числа операция сдвига будет учитывать только 4 младших бита. Это значит, что любой сдвиг на число, большее 15, будет эквивалентен сдвигу на остаток от деления этого числа на 16.

Таблица 1.6 – Операции языка программирования FVG-2024

|  |  |  |  |
| --- | --- | --- | --- |
| Операция | Примечание | Типы данных | Пример |
| ( | Приоритет операций | - | sum = (a + b) \* c; |
| ) |
| + | Суммирование | (short, short)  (char, char)  (uint, uint) | sum = a + b; |
| - | Вычитание | (short, short)  (uint, uint) | diff = a – b; |
| \* | Умножение | (short, short)  (uint, uint) | mul = a\*b; |
| : | Деление | (short, short)  (uint, uint) | div = a:b; |
| / | Сдвиг влево | (uint, uint) | pr = a / b; |
| \ | Сдвиг вправо | (uint, uint) | pr = a \ b; |
| = | Присваивание | (short, short)  (uint, uint)  (char, char)  (string, string) | sum = 15;  chr = ‘T’; |
| <,> | Знаки «больше», «меньше» для цикла | (uint, uint) | while(sum < diff) […]; |
| & | Оператор эквивалентности | (uint, uint) | while(sum & diff) […]; |
| ^ | Оператор неравенства | (uint, uint) | while(sum ^ diff) […]; |

Т.к. отрицательные числа не поддерживаются для uint и char, если результат операции меньше нуля, он вычитается из максимального значения.

## **Выражения и их вычисления**

Круглые скобки в выражении используются для изменения приоритета операций. Также не допускается запись двух подряд идущих арифметических операций. Выражение может содержать вызов функции, если эта функция уже содержится в стандартной библиотеке. Выражения вычисляются только после оператора присваивания.

## **Программные конструкции языка**

Ключевые программные конструкции языка программирования FVG-2024 представлены в таблице 1.7.

Таблица 1.7 – Программные конструкции языка FVG-2024

|  |  |
| --- | --- |
| Конструкция | Запись на языке FVG-2024 |
| Главная функция (точка входа) | main  {  …  return <идентификатор> / <литерал>;  } |
| Функция | var <тип данных> function <идентификатор> (<тип> <идентификатор>, …)  {… return <идентификатор> / <литерал>; }; |
| Цикл | while(a < 8)[ …]; |

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

## **Область видимости**

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

## **Семантические проверки**

Назначение семантического анализа – проверка смысловой правильности конструкций языка программирования. Таблица с перечнем семантических проверок, предусмотренных языком, приведена в таблице 1.8.

Таблица 1.8 – Семантические проверки

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Идентификаторы не должны повторно объявляться в пределах одной функции. |
| 2 | Тип возвращаемого значения должен совпадать с типом функции при её объявлении или подключении |
| 3 | Тип данных передаваемых значений в функцию должен совпадать с типом параметров при её объявлении или подключении |
| 4 | В функцию должно быть передано то число параметров, сколько ожидается |
| 5 | Тип данных результата выражения должен совпадать с типом данных идентификатора, которому оно присваивается |
| 6 | Типы данных операндов выражения должны быть одинаковыми |
| 7 | Тип данных string не может быть аргументом условной конструкции |
| 8 | Для типа char определены только операции + и - |
| 9 | Функции не должны подключаться дважды в пределах одной программы |

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

## **Распределение оперативной памяти на этапе выполнения**

Все переменные в языке программирования FVG-2024 размещаются в куче. Это означает, что управление памятью осуществляется динамически, предоставляя разработчику большую гибкость при создании и управлении объектами.

Куча представляет собой область оперативной памяти, предназначенную для хранения данных переменной длины или объектов, время жизни которых определяется во время выполнения программы. Такой подход обеспечивает эффективное использование памяти, позволяя выделять и освобождать ресурсы по мере необходимости.

Динамическое распределение памяти упрощает работу с большими объемами данных и сложными структурами, такими как массивы, списки или деревья. Однако важно учитывать, что работа с кучей требует осторожного обращения с памятью, чтобы избежать утечек или фрагментации памяти.

Для управления памятью в куче часто используется автоматический сборщик мусора или другие механизмы, предоставленные языком программирования, что позволяет разработчикам сосредоточиться на логике приложения, минимизируя ручное управление памятью.

## **Стандартная библиотека и её состав**

Стандартная библиотека FVG-2024 написана на высокоуровневом компилируемом языке программирования C++.

Для использования функций стандартной библиотеки, нужно явно подключить необходимую функцию с помощью ключевого слова make\_F, далее работа с ними производится как с пользовательскими функциями. Функции стандартной библиотеки с описанием представлены в таблице 1.9.

Таблица 1.9 – Состав стандартной библиотеки

|  |  |  |
| --- | --- | --- |
| Функция(C++) | Возвращаемое значение | Описание |
| uint ConvertCharToInt (char ch) | uint | Возвращает код символа |
| char ConvertIntToChar (uint number) | char | Возвращает символ с заданным кодом |
| uint WriteMonthNOW (uint number) | uint | Возвращает номер месяца |
| uint WriteDateNOW (uint number) | uint | Возвращает дату в формате ДДММГГГГ |
| uint GetHours(short number) | uint | Возвращает час |
| uint WriteMinutesNOW (short number) | uint | Возвращает минуты |
| char\* compareStr(const char\* str1, const char\* str2) | string | Возвращает значение лексикографического сравнения |

Функция compareStr(const char\* str1, const char\* str2) – является основной.

## **Ввод и вывод данных**

В языке FVG-2024 не реализованы средства ввода данных.

Для вывода данных в стандартный поток вывода предусмотрен оператор write, который базируется на приватных функциях стандартной библиотеки.

## **Точка входа**

В языке FVG-2024 каждая программа должна содержать главную функцию main — точку входа, с которой начинается последовательное выполнение программы. Эта функция служит стартовой точкой, где определяется начальная логика работы приложения и вызываются остальные функции.

## **Препроцессор**

Препроцессор в языке программирования FVG-2024 не предусмотрен.

## **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.

## **Объектный код**

FVG-2024 транслируется в язык ассемблера.

## **Классификация сообщений транслятора**

В случае возникновения ошибки в коде программы на языке FVG-2024 и выявления её транслятором в текущий файл протокола выводится сообщение. Классификация сообщений приведена в таблице 1.10.

Таблица 1.10. – Классификация сообщений транслятора

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

Компилятор может обрабатывать до 1000 различных ошибок.

## **Контрольный пример**

Код контрольного примера представлен в Приложении А.

# **2 Структура транслятора**

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

Транслятор преобразует программу, написанную на языке FVG-2024, в программу на языке ассемблера. Для указания выходных файлов используются параметры командной строки, описанные в соответствующем разделе. Основными компонентами транслятора являются лексический анализатор, синтаксический анализатор, семантический анализатор и генератор кода. Их взаимодействие представлено на рисунке 2.1.



Рисунок 2.1 – Структура транслятора

Лексический анализатор является первым этапом трансляции. Его задача — обнаружить ошибки, связанные с лексикой языка FVG-2024, и сформировать таблицы лексем и идентификаторов. На этом этапе входной текст программы разбивается на элементарные единицы (лексемы), которые проверяются на соответствие грамматике языка.

Синтаксический анализатор отвечает за распознавание конструкций программы. Используя таблицы лексем и идентификаторов, он формирует дерево разбора и промежуточное представление программы. Если обнаруживаются синтаксические ошибки, они фиксируются и выдаются пользователю для исправления.

На этапе семантического анализа проверяется корректность программы с точки зрения логики языка FVG-2024. Анализатор оценивает правильность использования переменных, типов данных, операций и других элементов языка. Все обнаруженные ошибки семантики сообщаются пользователю.

Генератор кода — завершающий этап трансляции. Он преобразует промежуточное представление программы, созданное на предыдущих этапах, в код на языке ассемблера. Входными данными для генератора служат таблицы идентификаторов, лексем и дерево разбора. Только программы, успешно прошедшие лексический, синтаксический и семантический анализы, могут быть переведены в ассемблерный код.

## **2.2 Перечень входных параметров транслятора**

Входные параметры представлены в таблице 2.2.

Таблица 2.2 – Входные параметры транслятора языка FVG-2024

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

Таблицы лексем и дерево разбора синтаксического анализатора выводятся в лог журнал.

## **2.3 Перечень протоколов, формируемых транслятором и их содержимое**

Таблица с перечнем протоколов, формируемых транслятором языка FVG-2024 и их назначением представлена в таблице 2.3.

Таблица 2.3 – Протоколы, формируемые транслятором языка FVG-2024

|  |  |
| --- | --- |
| Формируемый протокол | Описание протокола |
| Файл журнала, “\*.log ” | Является основным средством для фиксации результатов работы анализаторов и регистрации ошибок. В нем хранится краткая информация об исходном коде, что помогает пользователю отслеживать процесс компиляции и диагностики |
|  | возможных проблем. В частности, файл журнала содержит протокол работы лексического, синтаксического и семантического анализаторов, а также сообщения об обнаруженных ошибках. Такая структура файла позволяет пользователю оперативно выявлять и исправлять недостатки в коде. Следует отметить, что ошибки, связанные с открытием файла журнала или считыванием параметров, в него не записываются, поскольку они относятся к системным сбоям, обрабатываемым на более низком уровне. |
| “\*.asm” | Представляет собой результат работы транслятора. В этом файле хранится сгенерированный код на языке Ассемблера, который является финальной стадией преобразования исходного текста программы. Этот код предназначен для дальнейшей обработки ассемблером, компоновщиком или непосредственно для выполнения в виртуальной среде, поддерживающей соответствующую архитектуру. Наличие данного протокола позволяет пользователю получить представление о том, как исходный код был интерпретирован и преобразован транслятором. |

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

Формируемые протоколы обеспечивают не только контроль за процессом трансляции, но и прозрачность работы системы. Они помогают пользователю не только диагностировать ошибки, но и анализировать этапы преобразования программы, что способствует улучшению качества исходного кода.

# **3 Разработка лексического анализатора**

## **3.1 Структура лексического анализатора**

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



Рисунок 3.1 – Структура лексического анализатора

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

## **3.2 Контроль входных символов**

Исходный код на языке программирования FVG-2024 прежде чем транслироваться проверяется на допустимость символов. То есть изначально из входного файла считывается по одному символу и проверяется является ли он разрешённым.

Таблица для контроля входных символов представлена в приложении Б.

Принцип работы таблицы заключается в соответствии значения каждому элементу в шестнадцатеричной системе счисления значению в таблице ASCII.

Описание значения символов: T – разрешённый символ, F – запрещённый символ, S – пробельный символ, С – символ одинарной кавычки, L – символ-разделитель, D – символ двойной кавычки, O – символ начала комментария, N – символ новой строки.

## **3.3 Удаление избыточных символов**

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

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

Лексемы – это символы, соответствующие ключевым словам, символам операций и сепараторам, необходимые для упрощения дальнейшей обработки исходного кода программы. Данное соответствие описано в таблице 3.2.

Таблица 3.2 Соответствие ключевых слов, символов операций и сепараторов с лексемами

|  |  |  |  |
| --- | --- | --- | --- |
| Тип цепочки | Примечание | Цепочка | Лексема |
| Тип данных | Целочисленный знаковый тип данных | short | t |
|  | Целочисленный беззнаковый тип данных | uint | t |
| Строковый тип данных | string | t |
| Символьный тип данных | char | t |
| Лексема | Объявление переменной | var | d |
| Подключение функции библиотеки | make\_F | e |
| Оператор вывода | write | p |
| Объявление функции | function | f |
| Возврат значения из функции | return | r |
| Инструкция цикла | while | u |
| Блок инструкции цикла | [ | [ |
| ] | ] |
| Блок функции | { | { |
| } | } |
| Изменение приоритетности в выражении и отделение параметров функций | ( | ( |
| ) | ) |
| Сепараторы | ; | ; |
|  |  |  |  |

Окончание таблицы 3.2

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | , | , |
|  | Оператор присваивания | = | v |
|  | Условный оператор | < | b |
| > | b |
| & | b |
| ^ | b |
| Оператор | Знаки арифметических операций | + | v |
| -  -(унарный) | v  - |
| \* | V |
| / | v |
| \ | v |
| : | v |
| Идентификатор |  | [a-z|A-Z]+  [a-z|A-Z|0-9]\* | i |
| Литерал | Целочисленный литерал | [1-9]+[0-9]\* | l |
| Символьный литерал | [a-z|A-Z|0-9]\*  кроме ‘ | l |
| Строковый литерал | [a-z|A-Z|0-9]\* кроме ” | l |
| Точка входа |  | main | m |

Каждое выражение в языке FVG-2024 может быть представлено в виде детерминированного конечного автомата (ДКА). Такой автомат представляет собой модель с конечным числом состояний, которая используется для анализа и разбора выражения.

Для обработки выражения на соответствующий автомат из массива подается входная строка. Анализ выполняется с помощью регулярного выражения, которое описывает граф переходов данного автомата. Если анализ проходит успешно, разобранное выражение добавляется в таблицу лексем.

Если выражение представляет собой идентификатор или литерал, информация о нем дополнительно заносится в таблицу идентификаторов. Это позволяет хранить дополнительные данные, такие как тип идентификатора, значение литерала и другие свойства.

Пример конечных автоматов, соответствующих лексемам языка FVG-2024, представлены в приложении В.

## **3.5 Основные структуры данных**

Основные структуры таблиц лексем и идентификаторов данных языка FVG-2024, используемых для хранения, представлены в приложении Г.

В таблице лексем содержится лексема, её номер, полученный при разборе, номер строки в исходном коде, номер столбца в исходном коде, индекс таблицы идентификаторов (если нет соответствующего идентификатора, то индекс равен -1), а также специальное поле, в котором хранится значение лексемы.

В таблице идентификаторов содержится имя идентификатора, номер в таблице лексем, тип данных, тип идентификатора, его значение.

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

Индексы ошибок, обнаруживаемых лексическим анализатором, находятся в диапазоне 117-125. Также сам текст ошибки содержит в себе префикс [ЛЕКСИЧЕСКИЙ]. Перечень сообщений лексического анализатора представлен на рисунке 3.3.

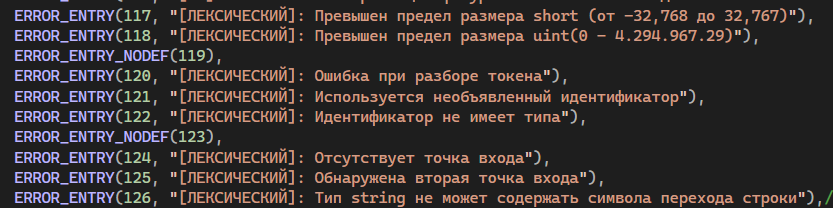


Рисунок 3.3 – Перечень ошибок лексического анализатора

## **3.7 Принцип обработки ошибок**

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

## **3.8 Параметры лексического анализатора**

Входные параметры используются для вывода результата работы всего транслятора. Они передаются аргументами через командную строку и рассмотрены в таблице 2.1

## **3.9 Алгоритм лексического анализа**

Алгоритм работы лексического анализа заключается в последовательном распознавании и разборе цепочек исходного кода и заполнение таблиц идентификаторов и лексем. Лексический анализатор производит распознаёт и разбирает цепочки исходного текста программы. Это основывается на работе конечных автоматов, которую можно представить в виде графов. В случае, если подходящий автомат не был обнаружен, запоминается номер строки, в которой находился этот токен и выводится сообщение об ошибке. Если токен разобран, то дальнейшие действия, которые будут с ним производиться, будут зависеть от того, чем он является. Регулярные выражения — аналитический или формульный способ задания регулярных языков. Они состоят из констант и операторов, которые определяют множества строк и множество операций над ними. Любое регулярное выражение можно представить в виде графа.

В случае, если токен является идентификатором, перед его именем записывается название функции, в которой он объявлен и после этого он заносится в таблицу идентификаторов.

В случае, если токен является идентификатором функции, название функции в которой он объявлен не записывается.

В случае, если токен является литералом, то он заносится в таблицу идентификаторов в виде abi, где a – имя функции, где объявлен литерал, b – “$LEX”, c –количество определённых литералов+1.

Когда встречаем токен, являющийся ключевым словом, которое отвечает за тип данных или вид идентификатора, заносим лексему, соответствующую ему, в таблицу лексем и запоминаем тип данных или вид идентификатора, которому он соответствует.

В последствии, когда встречаем идентификатор, заносим его в таблицу идентификаторов с соответствующим ему типом данных и видом идентификатора, и именем вида “ab”, где a – имя функции, где объявлен идентификатор, b – имя идентификатора.

Пример. Регулярное выражение для ключевого слова string: «string».

Граф конечного автомата для этой лексемы представлен на рисунке 3.3. S0 – начальное состояние, S6  – конечное состояние автомата.

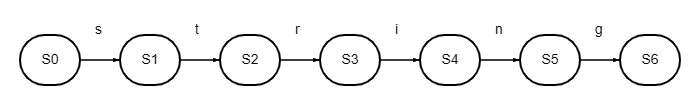


Рисунок 3.4 – Граф переходов для цепочки «string»

## **3.10 Контрольный пример**

Результат работы лексического анализатора – таблицы лексем и идентификаторов – представлен в приложении Г.

# **4 Разработка синтаксического анализатора**

## **Структура синтаксического анализатора**

Синтаксический анализ является следующим этапом трансляции, который выполняется после лексического анализа. Его основная задача — распознавание синтаксических конструкций языка FVG-2024. Входные данные для синтаксического анализатора: таблица лексем; таблица идентификаторов, сформированные на этапе лексического анализа. Выходом – дерево разбора. Структура синтаксического анализатора представлена на рисунке 4.1.



Рисунок 4.1 – Структура синтаксического анализатора FVG-2024

Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов. Выходной информацией – дерево разбора.

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

В синтаксическом анализаторе транслятора языка FVG-2024 используется контекстно-свободная грамматика [2], 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.

Грамматика языка FVG-2024 представлена в приложении Д.

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

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

Таблица 4.2 – Перечень правил и описание нетерминальных символов FVG-2024

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
|  |  |  |
| S | m{NrE;};  m{rE;};  dtfi(F){NrE;};S  dtfi(F){rE;};S  dtfi(){NrE;};S  dtfi(){rE;};S | Порождает правила, описывающее общую структуру программы |
| N | dti;  dti;N  ivE;  ivE;N  etfi(F);N  etfi(F);  pi;N  pi;  pl;N  p-l;N  pl;  p-l;  wi;N  wi;  wl;N  w-l;N  wl;  w-l;  u(B)[N];  u(B)[N];N | Порождает правила, описывающие конструкции языка |
| E | i  l и -l  (E)  i(W)  iM  lM  (E)M и Mvl и M-l  -E  i(W)M | Порождает правила, описывающие выражения |

Окончание таблицы 4.2

|  |  |  |
| --- | --- | --- |
| F | ti  ti,F | Порождает правила, описывающие параметры локальной функции при её объявлении |
| W | i  l  i,W  l,W | Порождает правила, описывающие принимаемые параметры функции |
| B | ibi  ibl  lbi  lbl | Порождает правила, описывающие условное выражение в операторе цикла |
| M | vE  v(E)  v(E)M  vEM | Порождает правила, описывающие знаки арифметических операций |

Протокол и ошибки работы синтаксического анализатора выводятся в лог журнал и в консоль.

## **4.3 Построение конечного магазинного автомата**

Конечный автомат с магазинной памятью представляет собой семерку [1], описание которой представлено в таблице 4.2. Структура данного автомата показана в приложении Д.

Таблица 4.3 – Описание компонентов магазинного автомата

|  |  |  |
| --- | --- | --- |
| Компонента | Определение | Описание |
|  | Множество состояний автомата | Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата |
|  | Алфавит входных символов | Алфавит является множеством терминальных и нетерминальных символов, описание которых содержится в разделе 1.2 и в таблице 4.1. |
|  | Алфавит специальных магазинных символов | Алфавит магазинных символов содержит стартовый символ и маркер дна стека |
|  | Функция переходов автомата | Функция представляет из себя множество правил грамматики, описанных в таблице 4.1. |

Окончание таблицы 4.3

|  |  |  |
| --- | --- | --- |
|  | Начальное состояние автомата | Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики (нетерминальный символ А) |
|  | Начальное состояние магазина автомата | Символ маркера дна стека ($) |
|  | Множество конечных состояний | Конечные состояние заставляют автомат прекратить свою работу. Конечным состоянием является пустой магазин автомата и совпадение позиции на входной ленте автомата с размером ленты |

Вывод результата работы синтаксического анализатора записан в log.

## **Основные структуры данных**

Основные структуры данных синтаксического анализатора включают в себя структуру магазинного конечного автомата и структуру грамматики Грейбах, описывающей правила языка FVG-2024. Данные структуры представлены в приложении Д.

## **4.5 Описание алгоритма синтаксического разбора**

Принцип работы автомата

1. В магазин помещается стартовый символ.
2. На основе ранее сформированных таблиц создается входная лента.
3. Автомат запускается для обработки входной ленты.
4. Если в магазине встречается нетерминальный символ, выбирается подходящая цепочка, соответствующая этому символу, и элементы этой цепочки записываются в магазин в обратном порядке.
5. Когда терминалы в стеке и ленте совпадают, терминал удаляется как из ленты, так и из стека. Если терминалы не совпадают, автомат возвращается в предыдущее сохраненное состояние и выбирает другую цепочку для текущего нетерминала.
6. Если в магазине вновь обнаруживается нетерминал, процесс повторяется с пункта 4.
7. Если стек становится пустым (достигнуто его дно), а входная лента полностью обработана, синтаксический анализ считается завершенным успешно, и формируется дерево разбора. Если лента не пуста или возникает другая ошибка, выбрасывается исключение.

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

Индексы ошибок, обнаруживаемых синтаксическим анализатором, находятся в диапазоне 600-609. Перечень сообщений синтаксического анализатора представлен на рисунке 4.4.

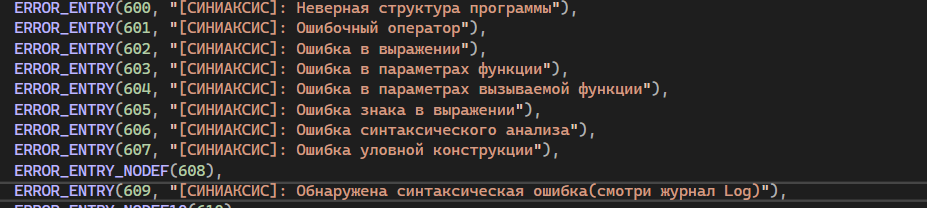


Рисунок 4.4 – Перечень сообщений синтаксического анализатора

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

Для вывода результата работы синтаксического анализатора используются входные параметры, описанные в пункте 2.2Перечень входных параметров транслятора в таблице 2.1.

## **4.8 Принцип обработки ошибок**

Процесс обработки ошибок синтаксическим анализатором осуществляется следующим образом:

1. Анализатор последовательно проверяет все правила и цепочки грамматики, пытаясь найти подходящее соответствие конструкции, указанной в таблице лексем.
2. Если подходящая цепочка не найдена, фиксируется ошибка синтаксического анализа.
3. При обнаружении ошибки в лог-файл записывается соответствующее сообщение, после чего работа компилятора прекращается

**4.9 Контрольный пример**

Пример разбора синтаксическим анализатором исходного кода на языке FVG-2024 представлен в приложении Е. Дерево разбора исходного кода также представлено в приложении Е.

# **Разработка семантического анализатора**

## **5.1 Структура семантического анализатора**

Семантический анализ происходит при выполнении фазы лексического анализа и реализуется в виде отдельных проверок текущих ситуаций в конкретных случаях: установки флага или нахождении в особом месте программы (оператор выхода из функции, оператор ветвления, вызов функции стандартной библиотеки). Структура семантического анализатора представлена на рисунке 5.1.



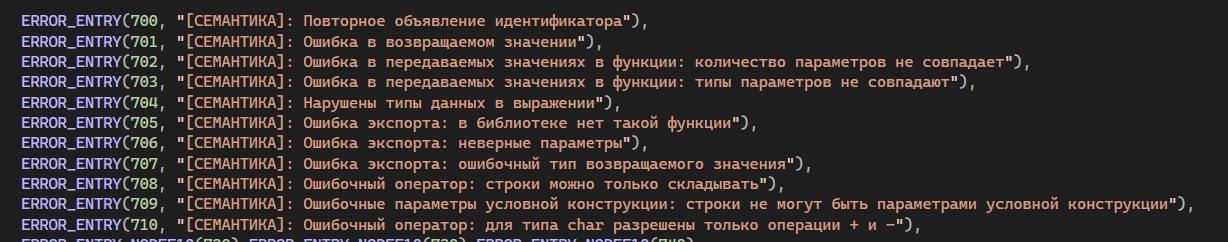
Рисунок 5.1 – Структура семантического анализатора

## **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**

Все ошибка семантического анализатора имеют идентификатор свыше 700. Сообщения, формируемые семантическим анализатором, представлены на рисунке 5.2.

Рисунок 5.2 – Перечень сообщений семантического анализатора

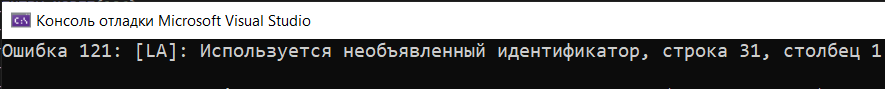
Текст семантической ошибки содержит в себе префикс [Semantic].

## **5.4 Принцип обработки ошибок**

При обнаружении хотя бы одной ошибки транслятор завершит свою работу c

Записью информации об ошибке в лог файл и в консоль.

Пример обнаружения ошибки лексическим анализатор и последующий вывод сообщения об ошибке в консоль и в log-файл представлен на рисунке 5.3



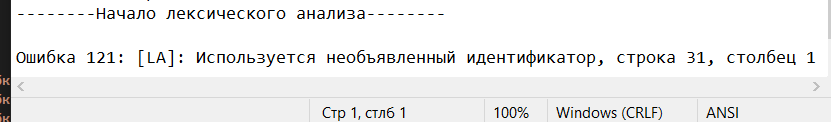


Рисунок 5.3 – Пример вывода информации об ошибке в консоль

## **5.5 Контрольный пример**

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

# **Преобразование выражений**

## **Выражения, допускаемые языком**

В языке FVG-2024 допускаются выражения, применимые к целочисленным типам данных. В выражениях поддерживаются арифметические операции, такие как +, -, \*, ^, :, /, \ и (), и вызовы функций как операнды арифметических выражений.

Приоритет операций представлен в таблице 6.1.

Таблица 6.1 – Приоритет операций в языке FVG-2024

|  |  |
| --- | --- |
| Операция | Значение приоритета |
| ( | 1 |
| ) | 1 |
| + | 2 |
| - | 2 |
| \* | 3 |
| : | 3 |
| / | 3 |
| \ | 3 |

Некоторые из операций таблицы 6.1 используются для типов, отличных от целочисленных.

## **Польская запись**

Обратная польская запись (ОПЗ) представляет собой форму записи математических и логических выражений, в которой операнды следуют перед знаками операций. Преимущество такого подхода заключается в устранении необходимости использования скобок.

Алгоритм преобразования выражений в ОПЗ:

1. Последовательно читаем символы исходной строки слева направо.
2. Если символ является идентификатором или литералом, добавляем его в выходную строку.
3. Если символ представляет собой функцию, помещаем его в стек.
4. Если встречается открывающая скобка, помещаем её в стек.
5. Если обнаружена закрывающая скобка, выталкиваем из стека в выходную строку все символы до ближайшей открывающей скобки. Открывающая и закрывающая скобки при этом удаляются.
6. Если символ представляет собой операцию, выталкиваем из стека в выходную строку все операции, имеющие более высокий или равный приоритет, после чего помещаем текущую операцию в стек.
7. Когда входная строка полностью обработана, переносим оставшиеся символы из стека в выходную строку.
8. Если идентификатор является именем функции, он заменяется на специальный символ «@».

## **6.3 Программная реализация обработки выражений**

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

## **6.4 Контрольный пример**

Таблица 6.2 – Пример преобразования выражения в обратную польскую запись

|  |  |  |
| --- | --- | --- |
| Исходная строка | Результирующая строка | Стек |
| a+b\*5/(c-2) |  |  |
| +b\*5/(c-2) | a |  |
| b\*5/(c-2) | a | + |
| \*5/(c-2) | ab | + |
| 5/(c-2) | ab | +\* |
| /(c-2) | ab | +\* |
| (c-2) | ab5\* | +/ |
| -2) | ab5\* | +/( |
| -2) | ab5\*c | +/( |
| 2) | ab5\*c | +/(- |
| ) | ab5\*c2 | +/(- |
|  | ab5\*c2- | +/ |
|  | ab5\*c2-/ | + |
|  | ab5\*c2-/+ |  |

Как результат успешного разбора, мы получаем пустой стек и заполненную результирующую строку.

# **Генерация кода**

## **7.1 Структура генератора кода**

Заключительным этапом трансляции языка FVG-2024 является генерация кода. Генерация объектного кода — это перевод компилятором внутреннего представления исходной программы в цепочку символов выходного языка. На вход генератора подаются таблицы лексем и идентификаторов, на основе которых генерируется файл с ассемблерным кодом.

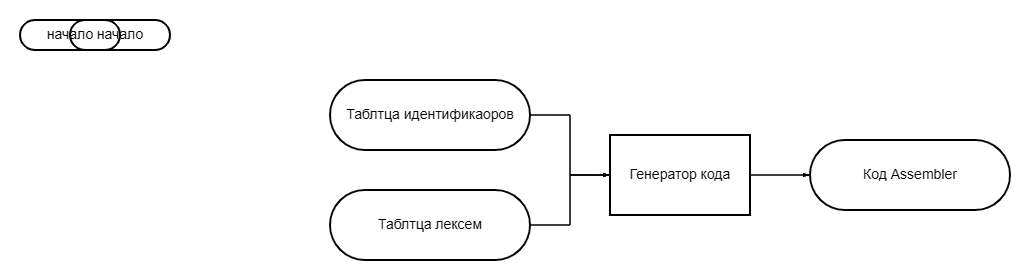


Рисунок 7.1 – Структура генератора кода

Таким образом, генератор кода выполняет не менее значимую часть компиляции.

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

Элементы таблицы идентификаторов расположены в разных сегментах языка ассемблера – .data и .const. Соответствия между типами данных идентификаторов на языке FVG-2024 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 Соответствия типов идентификаторов языка FVG-2024 и языка Ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке FVG-2024 | Тип идентификатора на языке ассемблера | Пояснение |
| short | SWORD | Хранит целочисленный тип данных со знаком. |
| char | BYTE | Хранит символьный тип данных. |
| string | BYTE | Хранит указатель на начало строки. |
| uint | DWORD | Хранит целочисленный тип данных без знака. |
| Лексема | SWORD  BYTE  BYTE  DWORD | Литералы: символьные,  целочисленные, строковые |

Идентификаторы языка FVG-2024 размещены в сегменте данных (.data). Литералы – в сегменте констант (.const).

## **7.3 Статическая библиотека**

В языке FVG-2024 предусмотрена поддержка статической библиотеки, содержащей функции, написанные на языке C++.

Вызовы стандартных функций доступны в тех же местах, что и вызовы пользовательских функций. Помимо этого, в стандартной библиотеке реализован набор функций для управления выводом, которые недоступны для прямого использования пользователем. Дополнительные функции приведены в таблице 7.2.

Таблица 7.2 - Дополнительные функции стандартной библиотеки

|  |  |  |
| --- | --- | --- |
| Функция(C++) | Возвращаемое значение | Описание |
| uint ConvertCharToInt (char ch) | uint | Возвращает код символа |
| char ConvertIntToChar (uint number) | char | Возвращает символ с заданным кодом |
| uint WriteMonthNOW (uint number) | uint | Возвращает номер месяца |
| uint WriteDateNOW (uint number) | uint | Возвращает дату в формате ДДММГГГГ |
| uint GetHours(short number) | uint | Возвращает час |
| uint WriteMinutesNOW (short number) | uint | Возвращает минуты |
| char\* compareStr(const char\* str1, const char\* str2) | string | Возвращает значение лексикографического сравнения |

Объявления функций из статической библиотеки генерируются автоматически.

## **7.4 Алгоритм работы генератора кода**

Алгоритм генерации кода выглядит следующим образом:

Генерирует заголовочную информацию (Листинг 7.3): в первую очередь -модель памяти, после чего - подключение библиотек, далее - прототипы внешних функций и сам размер стека.

|  |
| --- |
| .586  .model flat, stdcall  includelib libucrt.lib  includelib kernel32.lib  includelib ../FVG-2024/Debug/FVG-2024LIB.lib  ExitProcess PROTO :DWORD  ConvertIntToChar PROTO :DWORD  ConvertCharToInt PROTO :BYTE  compareStr PROTO :DWORD, :DWORD  WriteHoursNOW PROTO :DWORD  WriteMinutesNOW PROTO :DWORD  WriteMonthNOW PROTO :DWORD  WriteDateNOW PROTO :DWORD  writeUint PROTO :DWORD  writeShort PROTO :WORD  writeChar PROTO :BYTE  writeStr PROTO :DWORD  writelineUint PROTO :DWORD  writelineShort PROTO :WORD  writelineChar PROTO :BYTE  writelineStr PROTO :DWORD  .stack 4096 |

Листинг 7.3 –Заголовочная информация

Каждый литерал в поле .const соответствует идентификатору, зарегистрированному в таблице. Например, строковые литералы (обозначенные BYTE) используются для вывода сообщений об ошибках, описания выполнения программы или отображения данных пользователю.

Числовые литералы (DWORD) представляют целочисленные значения, такие как параметры функций, результат вычислений или начальные значения переменных. (Лист. 7.3).

Также можно заметить, что строки в кодировке ASCII заканчиваются нулевым символом (0), что используется для обозначения конца строки. Это позволяет работать с ними как с последовательностями символов, применяя стандартные функции обработки строк.

Структура .const обеспечивает упорядоченное хранение всех литералов программы, упрощая процесс генерации машинного кода и управление данными.

|  |
| --- |
| .const  num\_err byte 'Error: overflow', 0  divideOnZeroExeption BYTE " НА НОЛЬ ДЕЛИТЬ НЕЛЬЗЯ!!!.", 0 ;STR, вывод ошибки деления на ноль  MuituplyCounts$LEX1 DWORD 0 ;INT  MuituplyCounts$LEX2 DWORD 1 ;INT  main$LEX4 DWORD 6 ;INT  main$LEX5 DWORD 5 ;INT  main$LEX6 DWORD 3 ;INT  main$LEX7 BYTE "MuituplyCounts 6 \* 5 (3 Times) = ", 0 ;STR  main$LEX8 BYTE "Result (86 -> char) : ", 0 ;STR  main$LEX9 DWORD 86 ;INT  main$LEX10 BYTE "Result (V -> int) : ", 0 ;STR  main$LEX11 BYTE 'V' ;CHR  main$LEX12 BYTE "vania", 0 ;STR  main$LEX13 BYTE "Andrew", 0 ;STR  main$LEX14 SDWORD -5555 ;SHO  main$LEX15 DWORD 555 ;INT  main$LEX16 DWORD 111 ;INT  main$LEX17 BYTE "short Alice = ", 0 ;STR  main$LEX18 DWORD 8 ;INT  main$LEX19 DWORD 2 ;INT  main$LEX20 BYTE "(SHIFT LEFT) 8 / 2 = ", 0 ;STR  main$LEX22 BYTE "(SHIFT RIGHT) 8 \ 2 = ", 0 ;STR  main$LEX23 BYTE "Dear user TIME now is ...", 0 ;STR  main$LEX24 BYTE "|Hours| -- |Minutes| -- |Month| -- | Date |", 0 ;STR  main$LEX26 BYTE "| ", 0 ;STR  main$LEX27 BYTE " | | ", 0 ;STR  main$LEX30 BYTE " | | ", 0 ;STR  main$LEX31 BYTE " | ", 0 ;STR  main$LEX33 BYTE " |", 0 ;STR  main$LEX34 BYTE "|", 0 ;STR  main$LEX35 DWORD 0 ;INT |

Листинг 7.4 – Пример заполнения поля .const

Проходим таблицу идентификаторов и объявляем переменные в поле .data. (Листинг 7.5).

|  |
| --- |
| .data  MuituplyCountsfinal\_count DWORD 0 ;INT  mainresult\_M DWORD 0 ;INT  mainsi BYTE 0 ;CHR  maintemprorary DWORD 0 ;INT  mainvania DWORD 0 ;STR  mainAndrew DWORD 0 ;STR  mainAlice WORD 0 ;SHO  mainSHIFT DWORD 0 ;INT  mainSHIFT\_result1 DWORD 0 ;INT  mainSHIFT\_result2 DWORD 0 ;INT  mainFOR\_DATE\_FUNC DWORD 0 ;INT |

Листинг 7.5 – Пример заполнения поля .data

Для создания сегмента данных .code (см. Листинг 7.6) таблица идентификаторов просматривается с целью поиска функций. Найденные функции объявляются, и для каждой из них генерируется соответствующий код. Чтобы избежать совпадений с ключевыми словами ассемблера, перед именем функции добавляется символ «$». Во время генерации кода, при встрече оператора присваивания, выполняется разбор и описание вычисления выражения. Алгоритм преобразования выражений детально описан в пункте 7.3.

|  |
| --- |
| .code  $MuituplyCounts PROC uses ebx ecx edi esi , MuituplyCountsdigit1: DWORD , MuituplyCountsdigit2: DWORD , MuituplyCountstimes: DWORD  ; String #3 :ivl  push MuituplyCounts$LEX1  pop MuituplyCountsfinal\_count  While23Start:  mov eax, MuituplyCountstimes  mov ebx, MuituplyCounts$LEX2  cmp eax, ebx  jl While23End  ; String #6 :iviiivv  push MuituplyCountsfinal\_count  push MuituplyCountsdigit1  push MuituplyCountsdigit2  pop ebx  pop eax  mul ebx  push eax  pop ebx  pop eax  add eax, ebx  pop ebx  cmp eax, 32767  jg num\_error  cmp eax, -32767  jl num\_error  push eax  pop MuituplyCountsfinal\_count  ; String #7 :ivilv  push MuituplyCountstimes  push MuituplyCounts$LEX2  pop ebx  pop eax  sub eax, ebx  push eax  pop MuituplyCountstimes  jmp While23Start  While23End:  mov eax, MuituplyCountsfinal\_count  ret  $MuituplyCounts ENDP |

Листинг 7.6 – Пример заполнения поля .code

После генерации всех пользовательских функций, генерируется функция начала программы main в функции main по такому же принципу.

## **7.5 Контрольный пример**

Генерируемый код записывается в файл заданный параметром “-out”. Сгенерированный код можно посмотреть в приложении И.

# **8 Тестирование транслятора**

## **8.1 Тестирование фазы проверки на допустимость символов**

В языке FVG-2024 не разрешается использовать запрещённые входным алфавитом символы где-либо кроме строковых или символьных переменных. Результат использования запрещённого символа показан в таблице 8.1.

Таблица 8.1 – Тестирование фазы проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main{ПодОднимНебомХодим} | Ошибка 111: [IN]: Недопустимый символ в исходном файле (-in), строка 1, столбец 4 |

Запрещённые символы можно посмотреть в приложении Б.

## **8.2 Тестирование лексического анализатора**

На этапе лексического анализа могут возникнуть ошибки, описанные в пункте 3.7. Результаты тестирования лексического анализатора показаны в таблице 8.2.

Таблица 8.2 – Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main{  var uint 1bstu;  } | Ошибка 120: [ЛЕКСИЧЕСКИЙ]: Ошибка при разборе токена, строка 2, столбец 10 |
| main{  s = 5;  } | Ошибка 121: [ЛЕКСИЧЕСКИЙ]: Используется необъявленный идентификатор, строка 2, столбец 1 |
| var uint function F(uint a){  var answer;  } | Ошибка 122: [ЛЕКСИЧЕСКИЙ]: Идентификатор не имеет типа, строка 2, столбец 5 |
| var uint function a(){  var uint q;  } | Ошибка 124: [ЛЕКСИЧЕСКИЙ]: Отсутствует точка входа |
| main  {  return 0;  }  main  {  writeline "-------------- ";  return 0;  } | Ошибка 125: [ЛЕКСИЧЕСКИЙ]: Обнаружена вторая точка входа, строка 7, столбец 1 |

Ошибка лексического анализатора приводит к прекращению выполнения программы и записи соответствующей ошибки в лог журнал.

## **8.3 Тестирование синтаксического анализатора**

На этапе синтаксического анализа могут возникнуть ошибки, описанные в пункте 4.6. Результаты тестирования синтаксического анализатора показаны в таблице 8.3.

Таблица 8.3 – Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main{  var char adelina  } | Ошибка 609: [СИНИАКСИС]: Обнаружена синтаксическая ошибка(смотри журнал Log) |
| var uint function Func(uint a){  var uint answer;  answer = 1 \* 4;  return answer;  };  main  {  var uint factor;  factor = Func("w");  } | Ошибка 603: строка 9,[СИНИАКСИС]: Ошибка в параметрах функции |
| main  [  var uint factor;  factor = 4;  ] | Ошибка 600: строка 2,[СИНИАКСИС]: Неверная структура программы |
| var uint function Func(uint ; a){  return 5;  }; | Ошибка 604: строка 1,[СИНИАКСИС]: Ошибка в параметрах вызываемой функции |

Ошибка синтаксического анализатора также приводит к прекращению выполнения программы и записи соответствующей ошибки в лог журнал.

# **Заключение**

В ходе выполнения курсовой работы был разработан транслятор для языка программирования FVG-2024. Таким образом, были выполнены основные задачи данной курсовой работы:

* сформулирована спецификация языка FVG-2024;
* разработаны конечные автоматы и алгоритмы для реализация лексического анализатора;
* разработана контекстно-свободная, приведённая к ослабленной нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
* разработан семантический анализатор, осуществляющий проверку смысла используемых инструкций;
* разработан транслятор с языка программирования FVG-2024 на язык Assembler;
* проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка FVG-2024 включает:

* 4 типа данных;
* поддержку операции вывода;
* возможность вызова функций стандартной библиотеки;
* наличие 5 арифметических операторов для вычисления выражений;
* наличие 4 логических операторов для вычисления выражений;
* оператор цикла;

В результате работы были получены следующие результаты:

1. Было получено представление о структурах и процессах, используемых при построении трансляторов.
2. Были изучены основные различия и преимущества тех или иных средств трансляции.
3. Практический опыт работы с теорией языков программирования — разработка транслятора позволила глубже понять принципы работы с языками программирования, включая лексический, синтаксический и семантический анализ, а также их применение в реальных проектах.
4. Процесс тестирования и отладки — значительное внимание было уделено процессу тестирования и отладки всех компонентов транслятора, что позволило приобрести опыт в решении проблем, связанных с ошибками и оптимизацией работы программных систем.

# **Список использованных источников**

1. Янг М. и О'Рейли Т. Компиляторы на пальцах. Как писать код для программ, которые пишут код. 2015. — 384 с. [Электронный ресурс]. – Режим доступа: http://bourabai.ru/tpoi/compilers.htm.
2. Википедия: Обратная польская запись [Электронный ресурс]. – Режим доступа: https://en.wikipedia.org/wiki/Reverse\_Polish\_notation.
3. MASM для x86 [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/en-us/cpp/assembler/masm/masm-for-x64-ml64-exe?view=msvc-160>.
4. Ирвин К. Р. Язык ассемблера для процессоров Intel / К. Р. Ирвин. – M.: Вильямс, 2005. – 912с.

## ПРИЛОЖЕНИЕА

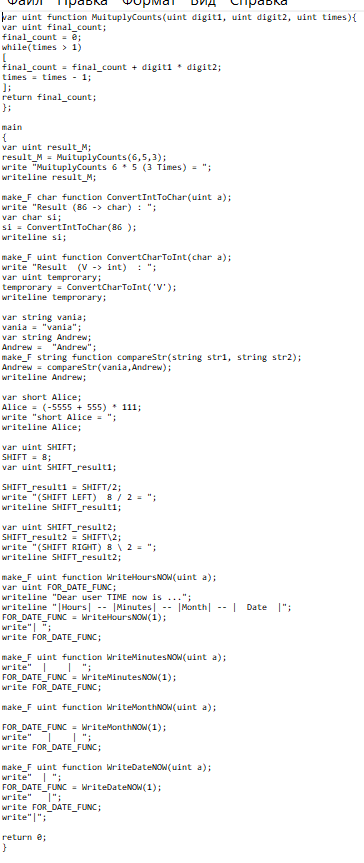


Рисунок 1 – Контрольный пример

## ПРИЛОЖЕНИЕ Б

|  |
| --- |
|  |

Рисунок 2 – Таблица входных символов

## ПРИЛОЖЕНИЕ В

|  |
| --- |
| FST l\_export(  str,  7,  NODE(1, RELATION('m', 1)),  NODE(1, RELATION('a', 2)),  NODE(1, RELATION('k', 3)),  NODE(1, RELATION('e', 4)),  NODE(1, RELATION('\_', 5)),  NODE(1, RELATION('F', 6)),  NODE()  );  FST l\_str(  str,  7,  NODE(1, RELATION('s', 1)),  NODE(1, RELATION('t', 2)),  NODE(1, RELATION('r', 3)),  NODE(1, RELATION('i', 4)),  NODE(1, RELATION('n', 5)),  NODE(1, RELATION('g', 6)),  NODE()  );  FST l\_string(  str,  5,  NODE(1, RELATION('c', 1)),  NODE(1, RELATION('h', 2)),  NODE(1, RELATION('a', 3)),  NODE(1, RELATION('r', 4)),  NODE()  );  FST l\_function(  str,  9,  NODE(1, RELATION('f', 1)),  NODE(1, RELATION('u', 2)),  NODE(1, RELATION('n', 3)),  NODE(1, RELATION('c', 4)),  NODE(1, RELATION('t', 5)),  NODE(1, RELATION('i', 6)),  NODE(1, RELATION('o', 7)),  NODE(1, RELATION('n', 8)),  NODE()  );  FST l\_var(  str,  4,  NODE(1, RELATION('v', 1)),  NODE(1, RELATION('a', 2)),  NODE(1, RELATION('r', 3)),  NODE()  );    FST l\_return(  str,  7,  NODE(1, RELATION('r', 1)),  NODE(1, RELATION('e', 2)),  NODE(1, RELATION('t', 3)),  NODE(1, RELATION('u', 4)),  NODE(1, RELATION('r', 5)),  NODE(1, RELATION('n', 6)),  NODE()  );      FST l\_write(  str,  6,  NODE(1, RELATION('w', 1)),  NODE(1, RELATION('r', 2)),  NODE(1, RELATION('i', 3)),  NODE(1, RELATION('t', 4)),  NODE(1, RELATION('e', 5)),  NODE()  );  FST l\_writeline(  str,  10,  NODE(1, RELATION('w', 1)),  NODE(1, RELATION('r', 2)),  NODE(1, RELATION('i', 3)),  NODE(1, RELATION('t', 4)),  NODE(1, RELATION('e', 5)),  NODE(1, RELATION('l', 6)),  NODE(1, RELATION('i', 7)),  NODE(1, RELATION('n', 8)),  NODE(1, RELATION('e', 9)),  NODE()  );  FST l\_main(  str,  5,  NODE(1, RELATION('m', 1)),  NODE(1, RELATION('a', 2)),  NODE(1, RELATION('i', 3)),  NODE(1, RELATION('n', 4)),  NODE()  );  FST l\_conditional(  str,  3,  NODE(1, RELATION('i', 1)),  NODE(1, RELATION('f', 2)),  NODE()  );  FST l\_semicolon(  str,  2,  NODE(1, RELATION(';', 1)),  NODE()  );  FST l\_comma(  str,  2,  NODE(1, RELATION(',', 1)),  NODE()  );  FST l\_braceleft(  str,  2,  NODE(1, RELATION('{', 1)),  NODE()  );  FST l\_braceright(  str,  2,  NODE(1, RELATION('}', 1)),  NODE()  );  FST l\_lefthesis(  str,  2,  NODE(1, RELATION('(', 1)),  NODE()  );  FST l\_cycleStart(  str,  2,  NODE(1, RELATION('[', 1)),  NODE()  );  FST l\_cycleEnd(  str,  2,  NODE(1, RELATION(']', 1)),  NODE()  );  FST l\_conditionalStart(  str,  2,  NODE(1, RELATION('[', 1)),  NODE()  );  FST l\_conditionalEnd(  str,  2,  NODE(1, RELATION(']', 1)),  NODE()  );  FST l\_righthesis(  str,  2,  NODE(1, RELATION(')', 1)),  NODE()  );  FST l\_verb(  str,  2,  NODE(8, RELATION('+', 1), RELATION('-', 1), RELATION('\*', 1),  RELATION('/', 1), RELATION(':', 1), RELATION('\\', 1), RELATION('%', 1), RELATION('=', 1)),  NODE()  );  FST l\_boolVerb(  str,  2,  NODE(4, RELATION('^', 1), RELATION('<', 1), RELATION('>', 1), RELATION('&', 1)),  NODE()  ) FST l\_itendificator(  str,  3,  NODE(106,  RELATION('a', 1),  RELATION('b', 1),  RELATION('c', 1),  RELATION('d', 1),  RELATION('e', 1),  RELATION('f', 1),  RELATION('g', 1),  RELATION('h', 1),  RELATION('i', 1),  RELATION('j', 1),  RELATION('k', 1),  RELATION('l', 1),  RELATION('m', 1),  RELATION('n', 1),  RELATION('o', 1),  RELATION('p', 1),  RELATION('q', 1),  RELATION('r', 1),  RELATION('s', 1),  RELATION('t', 1),  RELATION('u', 1),  RELATION('v', 1),  RELATION('w', 1),  RELATION('x', 1),  RELATION('y', 1),  RELATION('z', 1),  RELATION('A', 1),  RELATION('B', 1),  RELATION('C', 1),  RELATION('D', 1),  RELATION('E', 1),  RELATION('F', 1),  RELATION('G', 1),  RELATION('H', 1),  RELATION('I', 1),  RELATION('J', 1),  RELATION('K', 1),  RELATION('L', 1),  RELATION('M', 1),  RELATION('N', 1),  RELATION('O', 1),  RELATION('P', 1),  RELATION('Q', 1),  RELATION('R', 1),  RELATION('S', 1),  RELATION('T', 1),  RELATION('U', 1),  RELATION('V', 1),  RELATION('W', 1),  RELATION('X', 1),  RELATION('Y', 1),  RELATION('Z', 1),  RELATION('\_', 1),  RELATION('a', 2),  RELATION('b', 2),  RELATION('c', 2),  RELATION('d', 2),  RELATION('e', 2),  RELATION('f', 2),  RELATION('g', 2),  RELATION('h', 2),  RELATION('i', 2),  RELATION('j', 2),  RELATION('k', 2),  RELATION('l', 2),  RELATION('m', 2),  RELATION('n', 2),  RELATION('o', 2),  RELATION('p', 2),  RELATION('q', 2),  RELATION('r', 2),  RELATION('s', 2),  RELATION('t', 2),  RELATION('u', 2),  RELATION('v', 2),  RELATION('w', 2),  RELATION('x', 2),  RELATION('y', 2),  RELATION('z', 2),  RELATION('A', 2),  RELATION('B', 2),  RELATION('C', 2),  RELATION('D', 2),  RELATION('E', 2),  RELATION('F', 2),  RELATION('G', 2),  RELATION('H', 2),  RELATION('I', 2),  RELATION('J', 2),  RELATION('K', 2),  RELATION('L', 2),  RELATION('M', 2),  RELATION('N', 2),  RELATION('O', 2),  RELATION('P', 2),  RELATION('Q', 2),  RELATION('R', 2),  RELATION('S', 2),  RELATION('T', 2),  RELATION('U', 2),  RELATION('V', 2),  RELATION('W', 2),  RELATION('X', 2),  RELATION('Y', 2),  RELATION('Z', 2),  RELATION('\_', 2)  ),  NODE(126,  RELATION('a', 1),  RELATION('b', 1),  RELATION('c', 1),  RELATION('d', 1),  RELATION('e', 1),  RELATION('f', 1),  RELATION('g', 1),  RELATION('h', 1),  RELATION('i', 1),  RELATION('j', 1),  RELATION('k', 1),  RELATION('l', 1),  RELATION('m', 1),  RELATION('n', 1),  RELATION('o', 1),  RELATION('p', 1),  RELATION('q', 1),  RELATION('r', 1),  RELATION('s', 1),  RELATION('t', 1),  RELATION('u', 1),  RELATION('v', 1),  RELATION('w', 1),  RELATION('x', 1),  RELATION('y', 1),  RELATION('z', 1),  RELATION('A', 1),  RELATION('B', 1),  RELATION('C', 1),  RELATION('D', 1),  RELATION('E', 1),  RELATION('F', 1),  RELATION('G', 1),  RELATION('H', 1),  RELATION('I', 1),  RELATION('J', 1),  RELATION('K', 1),  RELATION('L', 1),  RELATION('M', 1),  RELATION('N', 1),  RELATION('O', 1),  RELATION('P', 1),  RELATION('Q', 1),  RELATION('R', 1),  RELATION('S', 1),  RELATION('T', 1),  RELATION('U', 1),  RELATION('V', 1),  RELATION('W', 1),  RELATION('X', 1),  RELATION('Y', 1),  RELATION('Z', 1),  RELATION('\_', 1),  RELATION('0', 1),  RELATION('1', 1),  RELATION('2', 1),  RELATION('3', 1),  RELATION('4', 1),  RELATION('5', 1),  RELATION('6', 1),  RELATION('7', 1),  RELATION('8', 1),  RELATION('9', 1),  RELATION('a', 2),  RELATION('b', 2),  RELATION('c', 2),  RELATION('d', 2),  RELATION('e', 2),  RELATION('f', 2),  RELATION('g', 2),  RELATION('h', 2),  RELATION('i', 2),  RELATION('j', 2),  RELATION('k', 2),  RELATION('l', 2),  RELATION('m', 2),  RELATION('n', 2),  RELATION('o', 2),  RELATION('p', 2),  RELATION('q', 2),  RELATION('r', 2),  RELATION('s', 2),  RELATION('t', 2),  RELATION('u', 2),  RELATION('v', 2),  RELATION('w', 2),  RELATION('x', 2),  RELATION('y', 2),  RELATION('z', 2),  RELATION('A', 2),  RELATION('B', 2),  RELATION('C', 2),  RELATION('D', 2),  RELATION('E', 2),  RELATION('F', 2),  RELATION('G', 2),  RELATION('H', 2),  RELATION('I', 2),  RELATION('J', 2),  RELATION('K', 2),  RELATION('L', 2),  RELATION('M', 2),  RELATION('N', 2),  RELATION('O', 2),  RELATION('P', 2),  RELATION('Q', 2),  RELATION('R', 2),  RELATION('S', 2),  RELATION('T', 2),  RELATION('U', 2),  RELATION('V', 2),  RELATION('W', 2),  RELATION('X', 2),  RELATION('Y', 2),  RELATION('Z', 2),  RELATION('\_', 2),  RELATION('0', 2),  RELATION('1', 2),  RELATION('2', 2),  RELATION('3', 2),  RELATION('4', 2),  RELATION('5', 2),  RELATION('6', 2),  RELATION('7', 2),  RELATION('8', 2),  RELATION('9', 2)  ),  NODE()  );; |

Листинг 1 – Таблица конечных автоматов

## ПРИЛОЖЕНИЕ Г

|  |
| --- |
|  |

Листинг 2 – Пример таблцы лексем и идентификаторов

## ПРИЛОЖЕНИЕ Д

|  |
| --- |
| Greibach greibach(NS('S'), TS('$'),  9,  Rule(NS('S'), GRB\_ERROR\_SERIES + 0,  6,  Rule::Chain(7, TS('m'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}')),//+  Rule::Chain(6, TS('m'), TS('{'), TS('r'), NS('E'), TS(';'), TS('}')),//+  Rule::Chain(15, TS('d'), TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';'), NS('S')),//+  Rule::Chain(14, TS('d'), TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';'), NS('S')),  Rule::Chain(14, TS('d'), TS('t'), TS('f'), TS('i'), TS('('), TS(')'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';'), NS('S')),  Rule::Chain(13, TS('d'), TS('t'), TS('f'), TS('i'), TS('('), TS(')'), TS('{'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';'), NS('S'))  ),  Rule(NS('N'), GRB\_ERROR\_SERIES + 1,  22,  Rule::Chain(4, TS('d'), TS('t'), TS('i'), TS(';')),//+  Rule::Chain(5, TS('d'), TS('t'), TS('i'), TS(';'), NS('N')), //+  Rule::Chain(4, TS('i'), TS('v'), NS('E'), TS(';')), //+  Rule::Chain(5, TS('i'), TS('v'), NS('E'), TS(';'), NS('N')),  Rule::Chain(9, TS('e'), TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS(';'), NS('N')), //+  Rule::Chain(8, TS('e'), TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS(';')),//+  Rule::Chain(8, TS('e'), TS('t'), TS('f'), TS('i'), TS('('), TS(')'), TS(';'), NS('N')),  Rule::Chain(7, TS('e'), TS('t'), TS('f'), TS('i'), TS('('), TS(')'), TS(';')),  Rule::Chain(4, TS('p'), TS('i'), TS(';'), NS('N')),//+  Rule::Chain(3, TS('p'), TS('i'), TS(';')),//+  Rule::Chain(4, TS('p'), TS('l'), TS(';'), NS('N')),//+  Rule::Chain(5, TS('p'), TS('-'), TS('l'), TS(';'), NS('N')),//+  Rule::Chain(3, TS('p'), TS('l'), TS(';')),//+  Rule::Chain(4, TS('p'), TS('-'), TS('l'), TS(';')),//+  Rule::Chain(4, TS('w'), TS('i'), TS(';'), NS('N')),//+  Rule::Chain(3, TS('w'), TS('i'), TS(';')),//+  Rule::Chain(4, TS('w'), TS('l'), TS(';'), NS('N')),//+  Rule::Chain(5, TS('w'), TS('-'), TS('l'), TS(';'), NS('N')),//+  Rule::Chain(3, TS('w'), TS('l'), TS(';')),//+  Rule::Chain(4, TS('w'), TS('-'), TS('l'), TS(';')),//+  Rule::Chain(8, TS('u'), TS('('), NS('B'), TS(')'), TS('['), NS('N'), TS(']'), TS(';')),  Rule::Chain(9, TS('u'), TS('('), NS('B'), TS(')'), TS('['), NS('N'), TS(']'), TS(';'), NS('N'))  ),  Rule(NS('E'), GRB\_ERROR\_SERIES + 2,  15,  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l')),  Rule::Chain(2, TS('-'), TS('l')),  Rule::Chain(3, TS('('), NS('E'), TS(')')),  Rule::Chain(4, TS('i'), TS('('), NS('W'), TS(')')),  Rule::Chain(2, TS('i'), NS('M')),  Rule::Chain(2, TS('l'), NS('M')),  Rule::Chain(3, TS('-'), TS('l'), NS('M')),  Rule::Chain(2, TS('-'), NS('E')),//+  Rule::Chain(4, TS('('), NS('E'), TS(')'), NS('M')),  Rule::Chain(3, NS('M'), TS('v'), TS('l')),//+  Rule::Chain(3, NS('M'), TS('-'), TS('l')),//+++++++++++++  Rule::Chain(5, TS('i'), TS('('), NS('W'), TS(')'), NS('M')),  Rule::Chain(3, TS('i'), TS('v'), NS('E')),//+  Rule::Chain(3, TS('i'), TS('-'), NS('E'))//+++++++++++++  ),  Rule(NS('M'), GRB\_ERROR\_SERIES + 3,  9,  Rule::Chain(2, TS('v'), NS('E')),//+  Rule::Chain(4, TS('v'), TS('('), NS('E'), TS(')')),  Rule::Chain(5, TS('v'), TS('('), NS('E'), TS(')'), NS('M')),  Rule::Chain(3, NS('M'), TS('v'), TS('l')),//+  Rule::Chain(3, TS('v'), NS('E'), NS('M')),  Rule::Chain(2, TS('-'), NS('E')),//+  Rule::Chain(4, TS('-'), TS('('), NS('E'), TS(')')),//+  Rule::Chain(5, TS('-'), TS('('), NS('E'), TS(')'), NS('M')),//+  Rule::Chain(3, TS('-'), NS('E'), NS('M'))//+  ),  Rule(NS('F'), GRB\_ERROR\_SERIES + 4,  2,  Rule::Chain(2, TS('t'), TS('i')),//+  Rule::Chain(4, TS('t'), TS('i'), TS(','), NS('F'))//+  ),  Rule(NS('W'), GRB\_ERROR\_SERIES + 5,  4,  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l')),  Rule::Chain(3, TS('i'), TS(','), NS('W')),  Rule::Chain(3, TS('l'), TS(','), NS('W'))  ),  Rule(NS('B'), GRB\_ERROR\_SERIES + 6,  3,  Rule::Chain(3, TS('i'), TS('b'), TS('i')),  Rule::Chain(3, TS('i'), TS('b'), TS('l')),  Rule::Chain(3, TS('l'), TS('b'), TS('i')),  Rule::Chain(3, TS('l'), TS('b'), TS('l'))  )  ); |

Листинг 3 – Пример таблцы лексем и идентификаторов

## ПРИЛОЖЕНИЕЕ

|  |
| --- |
| Шаг :Правило Входная лента Стек  0 :S->dtfi(F){NrE;};S dtfi(ti){dti;ivl;u(ibl)[i S$  1 : SAVESTATE: 1  1 : dtfi(ti){dti;ivl;u(ibl)[i dtfi(F){NrE;};S$  2 : tfi(ti){dti;ivl;u(ibl)[iv tfi(F){NrE;};S$  3 : fi(ti){dti;ivl;u(ibl)[ivi fi(F){NrE;};S$  4 : i(ti){dti;ivl;u(ibl)[iviv i(F){NrE;};S$  5 : (ti){dti;ivl;u(ibl)[ivivi (F){NrE;};S$  6 : ti){dti;ivl;u(ibl)[ivivi; F){NrE;};S$  7 :F->ti ti){dti;ivl;u(ibl)[ivivi; F){NrE;};S$  8 : SAVESTATE: 2  8 : ti){dti;ivl;u(ibl)[ivivi; ti){NrE;};S$  9 : i){dti;ivl;u(ibl)[ivivi;i i){NrE;};S$  10 : ){dti;ivl;u(ibl)[ivivi;iv ){NrE;};S$  11 : {dti;ivl;u(ibl)[ivivi;ivi {NrE;};S$  12 : dti;ivl;u(ibl)[ivivi;iviv NrE;};S$  13 :N->dti; dti;ivl;u(ibl)[ivivi;iviv NrE;};S$  14 : SAVESTATE: 3  14 : dti;ivl;u(ibl)[ivivi;iviv dti;rE;};S$  15 : ti;ivl;u(ibl)[ivivi;ivivl ti;rE;};S$  16 : i;ivl;u(ibl)[ivivi;ivivl; i;rE;};S$  17 : ;ivl;u(ibl)[ivivi;ivivl;] ;rE;};S$  18 : ivl;u(ibl)[ivivi;ivivl;]; rE;};S$  19 : TS\_NOK/NS\_NORULECHAIN  19 : RESSTATE  19 : dti;ivl;u(ibl)[ivivi;iviv NrE;};S$  20 :N->dti;N dti;ivl;u(ibl)[ivivi;iviv NrE;};S$  21 : SAVESTATE: 3  21 : dti;ivl;u(ibl)[ivivi;iviv dti;NrE;};S$  22 : ti;ivl;u(ibl)[ivivi;ivivl ti;NrE;};S$  23 : i;ivl;u(ibl)[ivivi;ivivl; i;NrE;};S$  24 : ;ivl;u(ibl)[ivivi;ivivl;] ;NrE;};S$  25 : ivl;u(ibl)[ivivi;ivivl;]; NrE;};S$  26 :N->ivE; ivl;u(ibl)[ivivi;ivivl;]; NrE;};S$  27 : SAVESTATE: 4  27 : ivl;u(ibl)[ivivi;ivivl;]; ivE;rE;};S$  28 : vl;u(ibl)[ivivi;ivivl;];r vE;rE;};S$  29 : l;u(ibl)[ivivi;ivivl;];ri E;rE;};S$  30 :E->l l;u(ibl)[ivivi;ivivl;];ri E;rE;};S$  31 : SAVESTATE: 5 |
|  |
| 0 :всего строк 335, синтаксический анализ выполнен без ошибок  0 : S->dtfi(F){NrE;};S  5 : F->ti,F  8 : F->ti,F  11 : F->ti  15 : N->dti;N  19 : N->ivE;N  21 : E->l  23 : N->u(B)[N];  25 : B->ibl  30 : N->ivE;N  32 : E->iM  33 : M->vE  34 : E->iM  35 : M->vE  36 : E->i  38 : N->ivE;  40 : E->iM  41 : M->vE  42 : E->l  47 : E->i  51 : S->m{NrE;}  53 : N->dti;N  57 : N->ivE;N  59 : E->i(W)  61 : W->l,W  63 : W->l,W  65 : W->l  68 : N->pl;N  71 : N->wi;N  74 : N->etfi(F);N  79 : F->ti  83 : N->pl;N  86 : N->dti;N  90 : N->ivE;N  92 : E->i(W)  94 : W->l  97 : N->wi;N  100 : N->etfi(F);N  105 : F->ti  109 : N->pl;N  112 : N->dti;N  116 : N->ivE;N  118 : E->i(W)  120 : W->l  123 : N->wi;N  126 : N->dti;N  130 : N->ivE;N  132 : E->l  134 : N->dti;N  138 : N->ivE;N  140 : E->l  142 : N->etfi(F);N  147 : F->ti,F  150 : F->ti  154 : N->ivE;N  156 : E->i(W)  158 : W->i,W  160 : W->i  163 : N->wi;N  166 : N->dti;N  170 : N->ivE;N  172 : E->(E)M  173 : E->-lM  175 : M->vE  176 : E->l  178 : M->vE  179 : E->l  181 : N->pl;N  184 : N->wi;N  187 : N->dti;N  191 : N->ivE;N  193 : E->l  195 : N->dti;N  199 : N->ivE;N  201 : E->iM  202 : M->vE  203 : E->l  205 : N->pl;N  208 : N->wi;N  211 : N->dti;N  215 : N->ivE;N  217 : E->iM  218 : M->vE  219 : E->l  221 : N->pl;N  224 : N->wi;N  227 : N->etfi(F);N  232 : F->ti  236 : N->dti;N  240 : N->wl;N  243 : N->wl;N  246 : N->ivE;N  248 : E->i(W)  250 : W->l  253 : N->pl;N  256 : N->pi;N  259 : N->etfi(F);N  264 : F->ti  268 : N->pl;N  271 : N->ivE;N  273 : E->i(W)  275 : W->l  278 : N->pi;N  281 : N->etfi(F);N  286 : F->ti  290 : N->ivE;N  292 : E->i(W)  294 : W->l  297 : N->pl;N  300 : N->pi;N  303 : N->etfi(F);N  308 : F->ti  312 : N->pl;N  315 : N->ivE;N  317 : E->i(W)  319 : W->l  322 : N->pl;N  325 : N->pi;N  328 : N->pl;  332 : E->l |

Листинг 4 – Пример Результата работы синтаксического анализатора и вывод дерева разбора

## ПРИЛОЖЕНИЕЗ

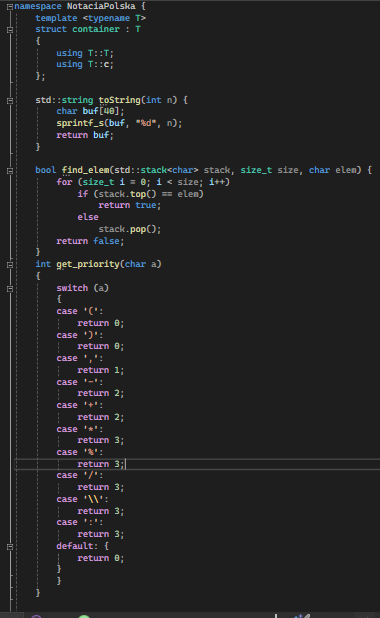


Рисунок 3 – Реализация алгоритма преобразования выражений к польской записи

## ПРИЛОЖЕНИЕ И

|  |
| --- |
| .586  .model flat, stdcall  includelib libucrt.lib  includelib kernel32.lib  includelib ../FVG-2024/Debug/FVG-2024LIB.lib  ExitProcess PROTO :DWORD  ConvertIntToChar PROTO :DWORD  ConvertCharToInt PROTO :BYTE  compareStr PROTO :DWORD, :DWORD  WriteHoursNOW PROTO :DWORD  WriteMinutesNOW PROTO :DWORD  WriteMonthNOW PROTO :DWORD  WriteDateNOW PROTO :DWORD  writeUint PROTO :DWORD  writeShort PROTO :WORD  writeChar PROTO :BYTE  writeStr PROTO :DWORD  writelineUint PROTO :DWORD  writelineShort PROTO :WORD  writelineChar PROTO :BYTE  writelineStr PROTO :DWORD  .stack 4096  .const  num\_err byte 'Error: overflow', 0  divideOnZeroExeption BYTE " НА НОЛЬ ДЕЛИТЬ НЕЛЬЗЯ!!!.", 0 ;STR, вывод ошибки деления на ноль  MuituplyCounts$LEX1 DWORD 0 ;INT  MuituplyCounts$LEX2 DWORD 1 ;INT  main$LEX4 DWORD 6 ;INT  main$LEX5 DWORD 5 ;INT  main$LEX6 DWORD 3 ;INT  main$LEX7 BYTE "MuituplyCounts 6 \* 5 (3 Times) = ", 0 ;STR  main$LEX8 BYTE "Result (86 -> char) : ", 0 ;STR  main$LEX9 DWORD 86 ;INT  main$LEX10 BYTE "Result (V -> int) : ", 0 ;STR  main$LEX11 BYTE 'V' ;CHR  main$LEX12 BYTE "vania", 0 ;STR  main$LEX13 BYTE "Andrew", 0 ;STR  main$LEX14 SDWORD -5555 ;SHO  main$LEX15 DWORD 555 ;INT  main$LEX16 DWORD 111 ;INT  main$LEX17 BYTE "short Alice = ", 0 ;STR  main$LEX18 DWORD 8 ;INT  main$LEX19 DWORD 2 ;INT  main$LEX20 BYTE "(SHIFT LEFT) 8 / 2 = ", 0 ;STR  main$LEX22 BYTE "(SHIFT RIGHT) 8 \ 2 = ", 0 ;STR  main$LEX23 BYTE "Dear user TIME now is ...", 0 ;STR  main$LEX24 BYTE "|Hours| -- |Minutes| -- |Month| -- | Date |", 0 ;STR  main$LEX26 BYTE "| ", 0 ;STR  main$LEX27 BYTE " | | ", 0 ;STR  main$LEX30 BYTE " | | ", 0 ;STR  main$LEX31 BYTE " | ", 0 ;STR  main$LEX33 BYTE " |", 0 ;STR  main$LEX34 BYTE "|", 0 ;STR  main$LEX35 DWORD 0 ;INT  .data  MuituplyCountsfinal\_count DWORD 0 ;INT  mainresult\_M DWORD 0 ;INT  mainsi BYTE 0 ;CHR  maintemprorary DWORD 0 ;INT  mainvania DWORD 0 ;STR  mainAndrew DWORD 0 ;STR  mainAlice WORD 0 ;SHO  mainSHIFT DWORD 0 ;INT  mainSHIFT\_result1 DWORD 0 ;INT  mainSHIFT\_result2 DWORD 0 ;INT  mainFOR\_DATE\_FUNC DWORD 0 ;INT  .code  $MuituplyCounts PROC uses ebx ecx edi esi , MuituplyCountsdigit1: DWORD , MuituplyCountsdigit2: DWORD , MuituplyCountstimes: DWORD  ; String #3 :ivl  push MuituplyCounts$LEX1  pop MuituplyCountsfinal\_count  While23Start:  mov eax, MuituplyCountstimes  mov ebx, MuituplyCounts$LEX2  cmp eax, ebx  jl While23End  ; String #6 :iviiivv  push MuituplyCountsfinal\_count  push MuituplyCountsdigit1  push MuituplyCountsdigit2  pop ebx  pop eax  mul ebx  push eax  pop ebx  pop eax  add eax, ebx  pop ebx  cmp eax, 32767  jg num\_error  cmp eax, -32767  jl num\_error  push eax  pop MuituplyCountsfinal\_count  ; String #7 :ivilv  push MuituplyCountstimes  push MuituplyCounts$LEX2  pop ebx  pop eax  sub eax, ebx  push eax  pop MuituplyCountstimes  jmp While23Start  While23End:  mov eax, MuituplyCountsfinal\_count  ret  $MuituplyCounts ENDP  main PROC  ; String #15 :ivilll@3!!  invoke $MuituplyCounts, main$LEX4, main$LEX5, main$LEX6  push eax ;результат функции  pop mainresult\_M  push offset main$LEX7  CALL writeStr  push mainresult\_M  CALL writelineUint  push offset main$LEX8  CALL writeStr  ; String #22 :ivil@1  invoke ConvertIntToChar, main$LEX9  push eax ;результат функции  pop eax  mov mainsi, al  push eax  movzx eax, mainsi  push eax  CALL writelineChar  pop eax  push offset main$LEX10  CALL writeStr  ; String #28 :ivil@1  invoke ConvertCharToInt, main$LEX11  push eax ;результат функции  pop maintemprorary  push maintemprorary  CALL writelineUint  ; String #32 :ivl  push offset main$LEX12  pop mainvania  ; String #34 :ivl  push offset main$LEX13  pop mainAndrew  ; String #36 :iviii@2!  invoke compareStr, mainvania, mainAndrew  push eax ;результат функции  pop mainAndrew  push mainAndrew  CALL writelineStr  ; String #40 :ivllvlv!!!  push main$LEX14  push main$LEX15  pop ebx  pop eax  add eax, ebx  pop ebx  cmp eax, 32767  jg num\_error  cmp eax, -32767  jl num\_error  push eax  push main$LEX16  pop ebx  pop eax  mul ebx  push eax  pop mainAlice  push offset main$LEX17  CALL writeStr  push mainAlice  CALL writelineShort  ; String #45 :ivl  push main$LEX18  pop mainSHIFT  ; String #48 :ivilv  push mainSHIFT  push main$LEX19  pop ebx  pop eax  push ecx ; сохраняем данные регистра ecx  mov ecx, ebx  SHL eax, cl  pop ecx  push eax  pop mainSHIFT\_result1  push offset main$LEX20  CALL writeStr  push mainSHIFT\_result1  CALL writelineUint  ; String #53 :ivilv  push mainSHIFT  push main$LEX19  pop ebx  pop eax  push ecx ; сохраняем данные регистра ecx  mov ecx, ebx  SHR eax, cl  pop ecx  push eax  pop mainSHIFT\_result2  push offset main$LEX22  CALL writeStr  push mainSHIFT\_result2  CALL writelineUint  push offset main$LEX23  CALL writelineStr  push offset main$LEX24  CALL writelineStr  ; String #61 :ivil@1  invoke WriteHoursNOW, MuituplyCounts$LEX2  push eax ;результат функции  pop mainFOR\_DATE\_FUNC  push offset main$LEX26  CALL writeStr  push mainFOR\_DATE\_FUNC  CALL writeUint  push offset main$LEX27  CALL writeStr  ; String #67 :ivil@1  invoke WriteMinutesNOW, MuituplyCounts$LEX2  push eax ;результат функции  pop mainFOR\_DATE\_FUNC  push mainFOR\_DATE\_FUNC  CALL writeUint  ; String #72 :ivil@1  invoke WriteMonthNOW, MuituplyCounts$LEX2  push eax ;результат функции  pop mainFOR\_DATE\_FUNC  push offset main$LEX30  CALL writeStr  push mainFOR\_DATE\_FUNC  CALL writeUint  push offset main$LEX31  CALL writeStr  ; String #78 :ivil@1  invoke WriteDateNOW, MuituplyCounts$LEX2  push eax ;результат функции  pop mainFOR\_DATE\_FUNC  push offset main$LEX33  CALL writeStr  push mainFOR\_DATE\_FUNC  CALL writeUint  push offset main$LEX34  CALL writeStr  mov eax, MuituplyCounts$LEX1  jmp endPoint  div\_by\_0:  push offset divideOnZeroExeption  CALL writelineStr  endPoint:  invoke ExitProcess, eax  main ENDP  num\_error::  push offset num\_err  call writelineStr  push - 1  call ExitProcess  end main |

Листинг 6 – Пример сгенерированного кода