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 MMA-2021»

Выполнил студент Мазько Максим Александрович

(Ф.И.О.)

Руководитель проекта асс. Пахолко Алёна Степановна

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

Заведующий кафедрой к.т.н., доц. Пацей Наталья Владимировна

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

Консультанты асс. Пахолко Алёна Степановна

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

Нормоконтролер асс. Пахолко Алёна Степановна

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

Курсовой проект защищен с оценкой

Минск 2021

**Содержание**

[**Введение 5**](#_Toc90871349)

[**1 Спецификация языка программирования 6**](#_Toc90871350)

[1.1 Характеристика языка программирования 6](#_Toc90871351)

[1.2 Алфавит языка 6](#_Toc90871352)

[1.3 Символы сепараторы 7](#_Toc90871353)

[1.4 Применяемые кодировки 7](#_Toc90871354)

[1.5 Типы данных 7](#_Toc90871355)

[1.6 Преобразование типов днных 8](#_Toc90871356)

[1.7 Идентификаторы 8](#_Toc90871357)

[1.8 Литералы 8](#_Toc90871358)

[1.9 Область видимости идентификаторов 9](#_Toc90871359)

[1.10 Инициализация данных 9](#_Toc90871360)

[1.11 Инструкции языка 9](#_Toc90871361)

[1.12 Операции языка 10](#_Toc90871362)

[1.13 Выражения и их вычисления 11](#_Toc90871363)

[1.14 Программные конструкции языка 11](#_Toc90871364)

[1.15 Область видимости 12](#_Toc90871365)

[1.16 Семантические проверки 12](#_Toc90871366)

[1.17 Распределение оперативной памяти на этапе выполнения 12](#_Toc90871367)

[1.18 Стандартная библиотека и её состав 12](#_Toc90871368)

[1.19 Ввод и вывод данных 13](#_Toc90871369)

[1.20 Точка входа 13](#_Toc90871370)

[1.21 Препроцессор 13](#_Toc90871371)

[1.22 Соглашения о вызовах 13](#_Toc90871372)

[1.23 Объектный код 13](#_Toc90871373)

[1.24 Классификация сообщений транслятора 13](#_Toc90871374)

[1.25 Контрольный пример 14](#_Toc90871375)

[**2 Структура транслятора 15**](#_Toc90871376)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 15](#_Toc90871377)

[2.2 Перечень входных параметров транслятора 16](#_Toc90871378)

[2.3 Перечень протоколов, формируемых транслятором и их содержимое 16](#_Toc90871379)

[**3 Разработка лексического анализатора 17**](#_Toc90871380)

[3.1 Структура лексического анализатора 17](#_Toc90871381)

[3.2 Контроль входных символов 17](#_Toc90871382)

[3.3 Удаление избыточных символов 18](#_Toc90871383)

[3.4 Перечень ключевых слов 18](#_Toc90871384)

[3.5 Основные структуры данных 19](#_Toc90871385)

[3.6 Принцип обработки ошибок 20](#_Toc90871386)

[3.7 Структура и перечень сообщений лексического анализатора 20](#_Toc90871387)

[3.8 Параметры лексического анализатора и режимы его работы 21](#_Toc90871388)

[3.9 Алгоритм лексического анализа 21](#_Toc90871389)

[3.10 Контрольный пример 22](#_Toc90871390)

[**4 Разработка синтаксического анализатора 23**](#_Toc90871391)

[4.1 Структура синтаксического анализатора 23](#_Toc90871392)

[4.2 Контекстно свободная грамматика, описывающая синтаксис языка 23](#_Toc90871393)

[4.3 Построение конечного магазинного автомата 25](#_Toc90871394)

[4.4 Основные структуры данных 26](#_Toc90871395)

[4.5 Описание алгоритма синтаксического разбора 26](#_Toc90871396)

[4.6 Структура и перечень сообщений синтаксического анализатора 26](#_Toc90871397)

[4.7 Параметры синтаксического анализатора и режимы его работы 27](#_Toc90871398)

[4.8 Принцип обработки ошибок 27](#_Toc90871399)

[4.9 Контрольный пример 28](#_Toc90871400)

[**5 Разработка семантического анализатора 29**](#_Toc90871401)

[5.1 Структура семантического анализатора 29](#_Toc90871402)

[5.2 Функции семантического анализатора 29](#_Toc90871403)

[5.3 Структура и перечень сообщений семантического анализатора 29](#_Toc90871404)

[5.4 Принцип обработки ошибок 30](#_Toc90871405)

[5.5 Контрольный пример 30](#_Toc90871406)

[**6 Преобразование выражений 31**](#_Toc90871407)

[6.1 Выражения, допускаемые языком 31](#_Toc90871408)

[6.2 Польская запись 31](#_Toc90871409)

[6.3 Программная реализация обработки выражений 32](#_Toc90871410)

[6.4 Контрольный пример 32](#_Toc90871411)

[**7 Генерация кода 33**](#_Toc90871412)

[7.1 Структура генератора кода 33](#_Toc90871413)

[7.2 Представление типов данных в оперативной памяти 33](#_Toc90871414)

[7.3 Статическая библиотека 33](#_Toc90871415)

[7.4 Особенности алгоритма генерации кода 34](#_Toc90871416)

[7.5 Входные параметры генератора кода 34](#_Toc90871417)

[7.6 Контрольный пример 34](#_Toc90871418)

[**8 Тестирование транслятора 35**](#_Toc90871419)

[8.1 Тестирование фазы проверки на допустимость символов 35](#_Toc90871420)

[8.2 Тестирование лексического анализатора 35](#_Toc90871421)

[8.3 Тестирование синтаксического анализатора 36](#_Toc90871422)

[8.4 Тестирование семантического анализатора 36](#_Toc90871423)

[Заключение 38](#_Toc90871424)

[**Список использованных источников 39**](#_Toc90871425)

[Приложение А 40](#_Toc90871426)

[Приложение Б 43](#_Toc90871427)

[Приложение В 45](#_Toc90871428)

[Приложение Г 47](#_Toc90871429)

[Приложение Д 51](#_Toc90871430)

[Приложение Е 53](#_Toc90871431)

# **Введение**

В данном курсовом проекте мы разрабатывали компилятор для собственного языка программирования.

Текст программы должен быть оттранслирован в соответствующую последовательность команд, прежде чем он может быть выполнен компьютером. Эта трансляция сама может быть описана программой. Транслирующая программа называется компилятором, а текст, который должен транслироваться, называется исходным текстом.

Транслятор MMA-2021 состоит из следующих частей:

– лексический анализатор;

– синтаксический анализатор;

– семантический анализатор;

– генератор исходного кода на языке ассемблера

В соответствии с курсовым проектом были определены следующие задачи:

– разработка спецификации языка программирования;

– разработка структуры транслятора;

– разработка программной реализации лексического анализатора;

– разработка программной реализации синтаксического анализатора;

– разработка программной реализации семантического анализатора;

– разработка программной реализации преобразования выражений;

– разработка программной реализации генератора кода;

– выполнить тестирование, разработанного программного обеспечения.

# **1 Спецификация языка программирования**

## **Характеристика языка программирования**

Язык MMA-2021 ­– это универсальный, строго типизированный, процедурный, компилируемый язык. Hе является объектно-ориентированным. В языке отсутствует преобразование типов. В языке поддерживается 2 типа данных: целочисленный беззнаковый (uint) и строковый (string). В стандартной библиотеке имеются функции для работы со строковым типом данных: strCmp – возвращает результат лексикографического сравнения строк, strLen – возвращает длину строки.

* 1. **Алфавит языка**

Алфавит языка MMA-2021 основан на кодировке Windows-1251, представленной на рисунке 1.1.



Рисунок 1.1 – Алфавит входных символов

Символы, используемые на этапе выполнения: [a…z], [A…Z], [0…9], [a…я], [А…Я], символы пробела, табуляции и перевода строки, спецсимволы: (){} , ; : + - / \* % > < ! ‘ .

## **Символы сепараторы**

Символы, которые являются сепараторами представлены в таблице 1.1.

Таблица 1.1 – Сепараторы

|  |  |
| --- | --- |
| Разделители | Назначение |
| ‘пробел’, ‘табуляция’,  ‘переход на новую строку’ | Разделяют входные лексемы. |
| +, -, \*, / | Арифметические операторы. Используются в арифметических операциях |
| = | Оператор присваивания. Используется для присваивания значения переменной |
| <, >, <=, >=, ==, != | Условные операторы. Используются для сравнения переменных и литералов |
| ( ) | Блок параметров функции, так же указывает приоритет в арифметических операциях |
| , | Разделяет параметры функции |
| { } | Блок функции или программной конструкции условного оператора |
| ; | Признак конца инструкции языка |
| ‘’ | Ограничивают строковый литерал |

## **Применяемые кодировки**

Для написания исходного кода на языке программирования MMA-2021 используется кодировка Windows-1251.

## **Типы данных**

В языке MMA-2021 есть 2 типа данных: целочисленный беззнаковый и строковый. Описание типов данных, предусмотренных в данным языке представлено в таблице 1.2.

Таблица 1.2 – Типы данных языка MMA-2021

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| Строковый тип данных **string** | Фундаментальный тип данных. Предусмотрен для объявления строк (1 символ – 1 байт). Максимальное количество символов в строке – 255.  Автоматическая инициализация строкой нулевой длины.  Возможные операции:  = – бинарный, присваивание значения. |

Продолжение таблицы 1.2

|  |  |
| --- | --- |
| Целочисленный беззнаковый тип данных **uint** | Фундаментальный тип данных. Предусмотрен для объявления целочисленных беззнаковых данных (4 байта). Предназначен для арифметических операций над числами.  Автоматически инициализируется нулевым значением.  Возможные операции:  + (бинарный) - суммирование;  - (бинарный) - вычитание;  \* (бинарный) - умножение;  / (бинарный) - деление;  = (бинарный) - присваивание значения;  В качестве операторов условия условного оператора поддерживаются следующие операторы:  > (бинарный) – оператор “больше”;  < (бинарный) – оператор “меньше”  != (бинарный) – оператор “не равно”  >= (бинарный) – оператор “больше либо равно”  <= (бинарный) – оператор “меньше либо равно”  == (бинарный) – оператор “сравнение” |

## **Преобразование типов днных**

В языке программирования MMA-2021 преобразование типов данных не поддерживается, т.е. язык является строго типизированным.

## **Идентификаторы**

Общее количество идентификаторов ограниченно максимальным размером таблицы идентификаторов (4096). Идентификаторы могут содержать символы как нижнего регистра, так и верхнего. Максимальная длина идентификатора равна 10 символам. Данные правила действуют для всех идентификаторов. Зарезервированные идентификаторы не предусмотрены. Идентификаторы не должны совпадать с ключевыми словами.

## **Литералы**

С помощью литералов осуществляется инициализация переменных. Все литералы являются rvalue. Имеются литералы целочисленные литералы десятичного и восьмеричного представления, а так же строковые литералы. Подробное описание литералов языка MMA-2021 представлены в таблице 1.3.

Таблица 1.3 – Описание литералов

|  |  |  |
| --- | --- | --- |
| Литералы | Пояснение | Пример |
| Целочисленные литералы в десятичном представлении | Последовательность цифр 0…9 с предшествующим знаком минус или без него (знак минус не отделяется пробелом) | new uint ch;  ch = 15;  15 – целочисленный литерал в десятичном представлении. |
| Целочисленные литералы в восьмеричном представлении | Последовательность цифр 0…7 с предшествующим знаком минус или без него (знак минус не отделяется пробелом), последующим символом “o” | new uint ch;  ch = 7o;  7o – целочисленный литерал в восьмеричном представлении. |
| Строковые литералы | Набор символов алфавита языка, заключенных в двойные кавычки | declare string str;  str = “Hello world”;  “Hello world” – строковый литерал. |

Ограничения на целочисленные литералы: не могут начинаться с 0, если их значение не 0; если литерал отрицательный, после знака “-” не может идти 0.

## **Область видимости идентификаторов**

Область видимости «сверху вниз» (по принципу С++). В языке MMA-2021 требуется обязательное объявление переменной перед её использованием. Все переменные должны находиться внутри программного блока языка. Имеется возможность объявления одинаковых переменных в разных блоках.

## **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 и литералы. При объявлении переменные инициализируются значением по умолчанию. Для **uint** значение 0, для **string** строка нулевой длины (‘’).

## **Инструкции языка**

Все возможные инструкции языка программирования MMA-2021 представлены в общем виде в таблице 1.5.

Таблица 1.5 – Инструкции языка программирования MMA-2021

|  |  |
| --- | --- |
| Инструкция | Запись на языке MMA-2021 |
| Объявление переменной | new <тип данных> <идентификатор>; |
| Присваивание | <идентификатор> = <выражение>;  Выражением может быть идентификатор, литерал, или вызов функции соответствующего типа. Для целочисленного типа выражение может быть дополнено арифметическими операциями с любым количеством операндов с использованием скобок. |
| Объявление внешней функции | <тип данных> function <идентификатор> (<тип данных> <идентификатор>, …) {…} |
| Точка входа | main  {  …  } |
| Возврат значения из подпрограммы | return <идентификатор> | <литерал>; |
| Вывод данных | output (<идентификатор> | <литерал>); |
| Условный оператор | if (<условие> | (<идентификатор> | <литерал>)  {  …  }  else  {  …  }  Блок else не обязателен. |

## **Операции языка**

Языком программирования MMA-2021 предусмотрены следующие операции, представленные в таблице 1.6.

Таблица 1.6 – Приоритетности операций языка программирования MMA-2021

|  |  |
| --- | --- |
| Операция | Приоритетность операции |
| (  ) | 1 |
| \*  /  % | 2 |
| +  - | 3 |
| ==  != | 4 |

Продолжение таблицы 1.6

|  |  |
| --- | --- |
| <  >  <=  >= | 5 |
| = | 6 |
| , | 7 |

Максимальным значением приоритетности является “1”, минимальным “7” соответственно.

## **Выражения и их вычисления**

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 целочисленное значение.

Перед генерацией кода каждое выражение приводится к записи в польской записи для удобства дальнейшего вычисления выражения на языке ассемблера. Преобразование выражений приведено в главе 5.

## **Программные конструкции языка**

Ключевые программные конструкции языка программирования MMA-2021 представлены в таблице 1.7.

Таблица 1.7 – Программные конструкции языка MMA-2021

|  |  |
| --- | --- |
| Конструкция | Описание |
| Главная функция (точка входа в приложение) | main  {  …  output <имя переменной/литерал>;  } |
| Функция | <тип> function <идентификатор>(<тип> <идентификатор>)  {  …  output <имя переменной/литерал>;  } |
| Блок | {  …  } |

Объявление функции допустимо только перед точкой входа в программу, так как иначе функции не будут входить в область видимости программы.

## **Область видимости**

В языке MMA-2021 переменные обязаны находиться внутри программного блока функций (по принципу С++). Объявление глобальных переменных не предусмотрено. Объявление пользовательских областей видимости не предусмотрено.

## **Семантические проверки**

Таблица с перечнем семантических проверок, предусмотренных языком MMA-2021, приведена в таблице 1.8.

Таблица 1.8 – Семантические проверки

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Вызов функции должен соответствовать её прототипу |
| 2 | Идентификатор должен быть объявлен до его использования |
| 3 | Операнды в арифметическом выражении не могут быть разных типов |
| 4 | Каждый идентификатор может быть объявлен только один раз |
| 5 | Проверка на превышение максимального размера строкового и целочисленного литералов |
| 6 | Соответствие типа возвращаемого значения с типом функции |
| 7 | Соответствие типов в выражениях |

Часть семантических проверок выполняется на этапе лексического анализа.

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

Переменные целочисленного типа находятся в стеке, так же в стеке находятся указатели на строки. Распределение оперативной памяти происходит на этапе генерации. Промежуточный код, таблица лексем и таблица идентификаторов сохраняются в структуры с выделенной под них динамической памятью, которая очищается по окончанию работы транслятора.

## **Стандартная библиотека и её состав**

Функции стандартной библиотеки с описанием представлены в таблице 1.9. Стандартная библиотека написана на языке программирования C++.

Таблица 1.9 – Состав стандартной библиотеки

|  |  |  |  |
| --- | --- | --- | --- |
| Имя функции | Возвращаемое значение | Принимаемые параметры | Описание |
| compare | uint | string x – строка  string y – строка | Функция лексикографически сравнивает строку x со строкой y |
| strln | uint | string x - строка | Функция вычисляет длину строки x |

Продолжение таблицы 1.9

|  |  |  |  |
| --- | --- | --- | --- |
| outInt | 0 | uint x - число | Функция выводит на консоль число x |
| outStr | 0 | string x - строка | Функция выводит на консоль строку x |

## **Ввод и вывод данных**

Ввод данных не поддерживается языком программирования MMA-2021.

output (<идентификатор или литерал>); – вывод в стандартный поток вывода. В зависимости от типа параметра определяется функция: outStr или outInt, которые входят в состав стандартной библиотеки и описаны в таблице 1.9.

## **Точка входа**

В языке MMA-2021 каждая программа должна содержать главную функцию (точку входа) main, с первой инструкции которой начинается последовательное выполнение команд программы.

## **Препроцессор**

Препроцессор в языке программирования MMA-2021 не предусмотрен.

## **Соглашения о вызовах**

В языке вызов функций происходит по соглашению о вызовах stdcall. Особенности stdcall:

– все параметры функции передаются через стек;

– память освобождает вызываемый код;

– занесение в стек параметров идёт справа налево.

## **Объектный код**

Язык программирования MMA-2021 транслируется в язык ассемблера, а затем в объектный код.

## **Классификация сообщений транслятора**

В случае возникновения ошибки в коде программы на языке MMA-2021 и выявления её транслятором в текущий файл протокола выводится сообщение. Их 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 |
| 120-200 | Ошибки лексического анализа |
| 300-399 | Ошибки семантического анализа |

Продолжение таблицы 1.10

|  |  |
| --- | --- |
| 600-699 | Ошибки синтаксического анализа |

## **Контрольный пример**

Контрольный пример представлен в приложении А.

# **2 Структура транслятора**

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

Транслятор преобразует программу, написанную на языке MMA-2021 в программу на языке ассемблера. Компонентами транслятора являются лексический, синтаксический и семантический анализаторы, а также генератор кода на язык ассемблера. Принцип их взаимодействия представлен на рисунке 2.1.

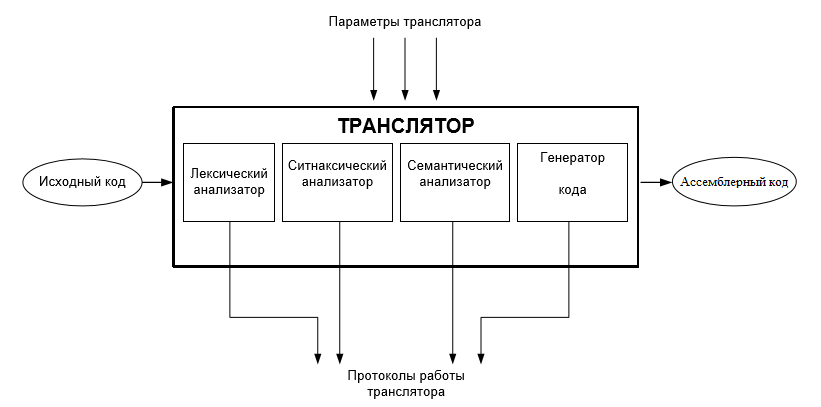


Рисунок 2.1 – Структура транслятора

Лексический анализ – первая фаза трансляции. Назначением лексического анализатора является нахождение ошибок лексики языка и формирование таблицы лексем и таблицы идентификаторов. Подробнее описан в 3 главе.

Семантический анализ в свою очередь является проверкой исходной программы на семантическую согласованность с определением языка, т.е. проверяет правильность текста исходной программы с точки зрения семантики. Подробное описание представлено в 5 главе.

В моём трансляторе часть функции семантического анализатора возложена на лексический анализатор.

Синтаксический анализ – это основная часть транслятора, предназначенная для распознавания синтаксических конструкций и формирования промежуточного кода. Входным параметром для синтаксического анализа является таблица лексем. Синтаксический анализатор распознаёт синтаксические конструкции, выявляет синтаксические ошибки при их наличии и формирует дерево разбора. Подробнее рассмотрен в главе 4.

Генератор кода – этап транслятора, выполняющий генерацию ассемблерного кода на основе полученных данных на предыдущих этапах трансляции.

Генератор кода принимает на вход таблицы идентификаторов и лексем и транслирует код на языке MMA-2021, прошедший все предыдущие этапы, в код на языке Ассемблера. Более полно описан в главе 7.

## **2.2 Перечень входных 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 MMA-2021

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<путь к in-файлу> | Входной файл с расширением .txt, в котором содержится исходный код на MMA-2021 | Не предусмотрено |
| -log:<путь к log-файлу> | Файл журнала для вывода протоколов работы программы. | <имя in-файла>.log |
| -out:<имя\_файла> | Выходной файл – результат работы транслятора. Содержит исходный код на языке асемблера. | <имя in-файла>.asm |
| -mfst:<имя\_файла> | Файл для записи результата синтаксического разбора | <имя in-файла>.mfst |

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

Таблица с перечнем протоколов, формируемых транслятором языка MMA-2021 и их назначением представлена в таблице 2.2

Таблица 2.2 – Протоколы, формируемые транслятором языка MMA-2021

|  |  |
| --- | --- |
| Формируемый протокол | Описание протокола |
| Файл журнала с параметром “-log:” | Содержит информацию о входных параметрах в приложение, о этапе проверки символов на допустимость, а также результат работы лексического анализатора. |
| Выходной файл с параметром “-out:” | Содержит сгенерированный код на языке Ассемблера. |
| Выходной файл с параметром “-mfst:” | Результат работы синтаксического анализа. Содержит правила разбора, а так же трассировку. |

# 

# **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 MMA-2021. На выходе формируется таблица лексем и таблица идентификаторов. Структура лексического анализатора представлена на рисунке 3.1



Рисунок 3.1 – Структура лексического анализатора MMA-2021

## **3.2 Контроль входных символов**

Таблица для контроля входных символов представлена на рисунке 3.2

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3.2. – Таблица контроля входных символов

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, I – игнорируемый символ, SPC – разделитель лексем, S – символ переноса строки, LX – символ литерала.

## **3.3 Удаление избыточных символов**

Избыточными символами являются символы табуляции, пробелы и переходы на новую строку.

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.

## **3.4 Перечень ключевых слов**

Лексический анализатор преобразует исходный текст, заменяя лексические единицы лексемами для создания промежуточного представления исходной программы. Соответствие токенов и лексем приведено в таблице 3.1.

Таблица 3.1 Соответствие ключевых слов, символов операций и сепараторов с лексемами

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Пояснение |
| uint, string | t | Названия типов данных языка. |
| Идентификатор | i | Длина идентификатора – 10 символов. |
| литерал | l | Литерал любого доступного типа. |
| function | f | Объявление функции. |
| return | r | Выход из функции/процедуры. |
| main | m | Главная функция. |
| new | n | Объявление переменной. |
| output | p | Вывод данных. |
| if | e | Указывает на начало условного блока. |
| else | c | Указывает на ложную ветвь условного блока. |
| ; | ; | Разделение выражений. |
| , | , | Разделение параметров функций. |
| { | } | Начало блока/тела функции. |
| } | } | Закрытие блока/тела функции. |
| ( | ( | Передача параметров в функцию, приоритет операций. |

Продолжение таблицы 3.1

|  |  |  |
| --- | --- | --- |
| ) | ) | Закрытие блока для передачи параметров, приоритет операций. |
| = | = | Знак присваивания |
| +  -  \*  / | +  -  \*  / | Знаки операций |
| >  <  <=  >=  ==  != | >  <  $  #  ~  & | Знаки логических операторов |

Каждому выражению соответствует регулярное выражение по которому происходит разбор данного выражения. На каждое регулярное выражение в массиве подаётся фраза и с помощью метода std::regex\_match, стандартной библиотеки <regex>, соответствующего данному регулярному выражению, происходит разбор. В случае успешного разбора выражения оно записывается в таблицу лексем. Если выражение является идентификатором или литералом, информация также заносится в таблицу идентификаторов. В приложении А находятся регулярные выражения, соответствующие лексемам языка MMA-2021.

## **3.5 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). Таблица идентификаторов содержит имя идентификатора (id), область видимости идентификатора (scope), номер в таблице лексем (idxfirstLE), тип данных (iddatatype), тип идентификатора (idtype) и значение (value). Код C++ со структурой таблицы лексем представлен на листинге 3.3. Код C++ со структурой таблицы идентификаторов представлен на листинге 3.4.

|  |
| --- |
| struct Entry  {  char lexema;  int sn;  int idxTI;  };  struct LexTable  {  int maxsize;  int size;  Entry\* table;  }; |

Листинг 3.3 – Код структуры таблицы лексем

|  |
| --- |
| enum class IDDATATYPE { UINT = 1, STR = 2, BOOL = 3 };  enum class IDTYPE { V = 1, F = 2, P = 3, L = 4 };  struct Entry  {  int idxfirstLE;  std::string id;  std::string scope;  IDDATATYPE iddatatype;  IDTYPE idtype;  union  {  int vint;  struct  {  char str[TI\_STR\_MAXSIZE - 1];  int len;  } vstr;  } value;  };  struct IdTable  {  int maxsize;  int size;  Entry\* table;  }; |

Листинг 3.4 – Код структуры таблицы идентификаторов

## **3.6 Принцип обработки ошибок**

При возникновении критической ошибки – работа транслятора прекращается.

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

Перечень сообщений лексического анализатора представлен на листинге 3.4.

|  |
| --- |
| ERROR\_ENTRY(120, "[Лексическая] Превышен максимальный размер таблицы лексем"),  ERROR\_ENTRY(121, "[Лексическая] Таблица лексем переполнена"),  ERROR\_ENTRY(122, "[Лексическая] Нераспознаянная лексема"),  ERROR\_ENTRY(123, "[Лексическая] Выход за пределы таблицы лексем"),  ERROR\_ENTRY(124, "[Лексическая] Превышена длина строковго литерала"),  ERROR\_ENTRY(125, "[Лексическая] Превышен максимальный размер таблицы идентификаторов"),  ERROR\_ENTRY(126, "[Лексическая] Таблица идентификаторов переполнена"),  ERROR\_ENTRY(127, "[Лексическая] Выход за пределы таблицы идентификаторов"),  ERROR\_ENTRY(128, "[Лексическая] Повторное объявление main"),  ERROR\_ENTRY(129, "[Лексическая] Отсутствие входной точки программы функции main"),  ERROR\_ENTRY(130, "[Лексическая] Превышено значение целочисленного литерала"),  ERROR\_ENTRY(131, "[Лексическая] Необъявленный идентификатор"),  ERROR\_ENTRY(132, "[Лексическая] Незакрытый строковый литерал"),  ERROR\_ENTRY(133, "[Лексическая] Превышена длина идентификатора"), |

Листинг 3.4 – Перечень ошибок лексического анализатора

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

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

## **3.9 Алгоритм лексического анализа**

Лексический анализ выполняется программой (входящей в состав транслятора), называемой лексическим анализатором. Цель лексического анализа — выделение и классификация лексем в тексте исходной программы. Лексический анализатор распознаёт и разбирает цепочки исходного текста программы. Этот разбор основывается на работе конечных автоматов, которую можно представить в виде графов.

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.

Пример. Регулярное выражение для ключевого слова num: ‘uint’.

Граф конечного автомата для этой лексемы представлен на рисунке 3.5. S0 – начальное состояние, S4 – конечное состояние автомата.

Изображение выглядит как текст, спортивная игра, спорт

Автоматически созданное описание

Рисунок 3.5 – Граф переходов для цепочки ‘uint’

## **3.10 Контрольный пример**

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

# **4 Разработка синтаксического анализатора**

## **4.1 Структура синтаксического анализатора**

Синтаксический анализ – это фаза трансляции, выполняемая после лексического анализа и предназначенная для распознавания синтаксических конструкций. Входом для синтаксического анализа является таблица лексем и таблица идентификаторов, полученные после фазы лексического анализа. Выходом – дерево разбора.

Структура синтаксического анализатора представлена на рисунке 4.1.

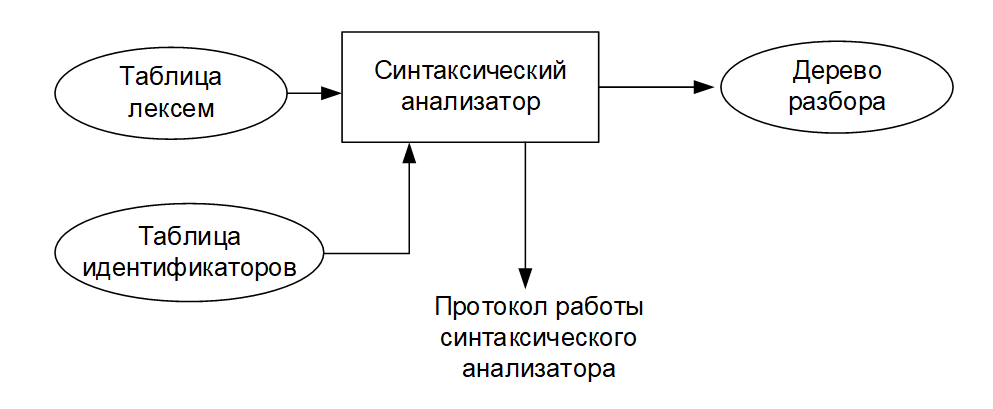


Рисунок 4.1 – Структура синтаксического анализатора

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

В синтаксическом анализаторе транслятора языка MMA-2021 используется контекстно-свободная грамматика типа II в иерархии Хомского (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.

Грамматика языка MMA-2021 представлена в приложении Б.

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

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

Таблица 4.1 – Перечень правил, составляющих грамматику языка и описание нетерминальных символов MMA-2021

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Какие правила порождает |
| S | S->tfiPTS  S->mT  S->dtfiP;S | Стартовые правила, описывающие общую структуру программы |
| T | T->{rV;}  T->{KrV;} | Привила для тела функций |
| P | P->(E)  P->() | Правила для параметров объявляемых функций |
| E | E->ti  E->ti, E | Правила для списка параметров функции |
| F | F->(N)  F->() | Правила для вызова функций |
| N | N->i  N->l  N->i, N  N->l, N | Правила для параметров вызываемой функции |
| Z | Z->(iLi)  Z->(iLl)  Z->(lLi)  Z->(lLl)  Z->(V) | Правила для условного оператора |
| L | L-><  L->>  L->$  L->#  L->~  L->& | Правила для логических операторов |
| A | A->+  A->-  A->\*  A->/  A->% | Правила для арифметических операторов |
| V | V->i  V->l | Правила для простых выражений |
| W | W->l  W->i  W->(W)  W->(W)AW  W->iF  W->iAW | Правила для сложных выражений |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| W | W->l  W->i  W->(W)  W->(W)AW  W->iF  W->iAW | Правила для сложных выражений |
| K | K->dti=W;K  K->i=W;  K->dti;K  K->i=W;K  K->pV;K  K->{K}  K->{K}K  K->eZ{K}K  K->eZ{K}y{K}K  K->iF;K  K->dti=W;  K->dti;  K->pV;  K->eZ{K}  K->eZ{K}y{K}  K->iF; | Правила для синтаксических конструкций |

## **4.3 Построение конечного магазинного автомата**

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

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 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 |

Продолжение таблицы 4.2

|  |  |  |
| --- | --- | --- |
|  | 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 (нетерминальный символ S) |
|  | 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 включают в себя структуру магазинного автомата, выполняющего разбор исходной ленты, и структуры грамматики Грейбах, описывающей синтаксические правила языка MMA-2021. Данные структуры представлены в приложении В.

## **4.5 Описание алгоритма синтаксического разбора**

Принцип работы автомата следующий:

1. В магазин записывается стартовый символ грамматики;
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 не терминала;
6. Если в магазине встретился не терминал, 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 Структура и перечень сообщений синтаксического анализатора**

Перечень сообщений синтаксического анализатора представлен на листинге 4.1.

|  |
| --- |
| ERROR\_ENTRY(600, "[Синтаксическая] Неверная структура программы"),  ERROR\_ENTRY(601, "[Синтаксическая] Не найден список параметров функции"),  ERROR\_ENTRY(602, "[Синтаксическая] Ошибка в теле функции"),  ERROR\_ENTRY(603, "[Синтаксическая] Ошибка в теле процедуры"),  ERROR\_ENTRY(604, "[Синтаксическая] Ошибка в списке параметров функции"),  ERROR\_ENTRY(605, "[Синтаксическая] Ошибка в вызове функции/выражении"),  ERROR\_ENTRY(606, "[Синтаксическая] Ошибка в списке фактических параметров функции"),  ERROR\_ENTRY(607, "[Синтаксическая] Ошибка в условии условного выражения"),  ERROR\_ENTRY(608, "[Синтаксическая] Неверный условный оператор"),  ERROR\_ENTRY(609, "[Синтаксическая] Неверный арифметический оператор"),  ERROR\_ENTRY(610, "[Синтаксическая] Неверное выражение. Ожидаются только идентификаторы/литералы"),  ERROR\_ENTRY(611, "[Синтаксическая] Ошибка в арифметическом выражении"),  ERROR\_ENTRY(612, "[Синтаксическая] Недопустимая синтаксическая конструкция"),  ERROR\_ENTRY(613, "[Синтаксическая] Синтаксический анализ завершён с ошибкой"), |

Листинг 4.1 – Перечень сообщений синтаксического анализатора

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

Входным параметром синтаксического анализатора является таблица лексем, полученная на этапе лексического анализа, а также правила контекстно-свободной грамматики в форме Грейбах.

Выходными параметрами являются трассировка прохода таблицы лексем (при наличии разрешающего ключа) и правила разбора, которые записываются в файл протокола данного этапа обработки.

## **4.8 Принцип обработки ошибок**

Обработка ошибок происходит следующим образом:

1. Синтаксический анализатор перебирает все правила и цепочки правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем.
2. Если невозможно подобрать подходящую цепочку, то генерируется соответствующая ошибка.
3. Все ошибки записываются в общую структуру ошибок.
4. В случае нахождения ошибки, после всей процедуры трассировки в протокол будет выведено диагностическое сообщение.

В структуре грамматики Грейбах цепочки в правилах расположены в порядке приоритета, самые часто используемые располагаются выше, а те, что используются реже – ниже.

## **4.9 Контрольный пример**

Пример разбора синтаксическим анализатором исходного кода на языке MMA-2021 представлен в приложении Г. Дерево разбора исходного кода также представлено в приложении Г

# **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.

Изображение выглядит как текст

Автоматически созданное описание

## **5.2 Функции семантического анализатора**

Семантический анализатор проверяет правильность составления программных конструкций. При невозможности подобрать правило перехода будет выведен код ошибки, а так же код этой ошибки. Информация об ошибках выводится в консоль, а так же в протокол работы.

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

Сообщения, формируемые семантическим анализатором, представлены на листинге 5.1.

|  |
| --- |
| ERROR\_ENTRY(300, "[Cемантическая] Функция main должна возвращать числовое значение"),  ERROR\_ENTRY(301, "[Cемантическая] Тип функции и тип возвращаемого значения отличаются"),  ERROR\_ENTRY(302, "[Cемантическая] Несоответствие типов параметров в вызываемой функции"),  ERROR\_ENTRY(303, "[Cемантическая] Неверное количество параметров вызываемой функции"),  ERROR\_ENTRY(304, "[Cемантическая] Несоответсвие присваимаего типа данных"),  ERROR\_ENTRY(305, "[Cемантическая] Использование имени переменной в качестве функции"),  ERROR\_ENTRY(306, "[Cемантическая] Попытка переопределения идентификатора"),  ERROR\_ENTRY(307, "[Cемантическая] Недопустимое значение переменной типа UINT"),  ERROR\_ENTRY(308, "[Cемантическая] Недопустимая операция над данным типом"),  ERROR\_ENTRY(309, "[Семантическая] Неверное условие в условном операторе"),  ERROR\_ENTRY(310, "[Семантическая] Недопустимый возврат функции типы UINT"), |

Листинг 5.1 – Перечень сообщений семантического анализатора

## **5.4 Принцип обработки ошибок**

При обнаружении ошибки в исходном коде программы семантический анализатор формирует сообщение об ошибки и выводит его в файл с протоколом работы, заданный параметром –log:.

## **5.5 Контрольный пример**

Результат работы контрольного примера расположен в приложении А, где показан результат лексического анализатора, т.к. представленные таблицы лексем и идентификаторов проходят лексическую и часть семантических проверок одновременно.

# **6 Преобразование выражений**

## **6.1 Выражения, допускаемые языком**

В языке MMA-2021 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 MMA-2021

|  |  |
| --- | --- |
| Приоритет | Операция |
| 0 | ( |
| 0 | ) |
| 1 | , |
| 2 | + |
| 2 | - |
| 3 | \* |
| 3 | / |
| 3 | % |

Скобки, в зависимости от их применения, могут иметь разный приоритет, либо 0, либо 4.

## **6.2 Польская запись**

Выражения в языке MMA-2021 преобразовываются к обратной польской записи.

Польская запись – это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок.

Обратная польская запись – это форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. 

Алгоритм построения:

– исходная строка: выражение;

– результирующая строка: польская запись;

– стек: пустой;

– результирующая строка: польская запись;

– исходная строка просматривается слева направо;

– операнды переносятся в результирующую строку в порядке их следования;

– операция записывается в стек, если стек пуст или в вершине стека лежит отрывающая скобка;

– операция выталкивает все операции с большим или равным приоритетом в результирующую строку;

– запятая не помещается в стек, если в стеке операции, то все выбираются в строку;

– отрывающая скобка помещается в стек;

– закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются;

– закрывающая скобка с приоритетом, равным 4, выталкивает все до открывающей с таким же приоритетом и генерирует @ – специальный символ, в которого записывается информация о вызываемой функции, а в поле приоритета для данной лексемы записывается число параметров вызываемой функции;

– по концу разбора исходной строки все операции, оставшиеся в стеке, выталкиваются в результирующую строку.

Таблица 6.2 – Пример преобразования выражения в обратную польскую запись

|  |  |  |
| --- | --- | --- |
| Исходная строка | Результирующая строка | Стек |
| b\*2 - i(l) |  |  |
| \*2 - i(l) | b |  |
| 2 - i(l) | b | \* |
| - u(l) | b2 | \* |
| u(l) | b2\* | - |
| (l) | b2\* | - |
| l) | b2\* | - |
| ) | b2\*i | - |
|  | b2\*i@1- |  |

Использование польской записи позволяет вычислить выражение за один проход.

## **6.3 Программная реализация обработки выражений**

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

## **6.4 Контрольный пример**

В приложении Д приведены изменённые таблицы лексем и идентификаторов, отображающие результаты преобразования выражений в польский формат.

# **7 Генерация кода**

## **7.1 Структура генератора 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.

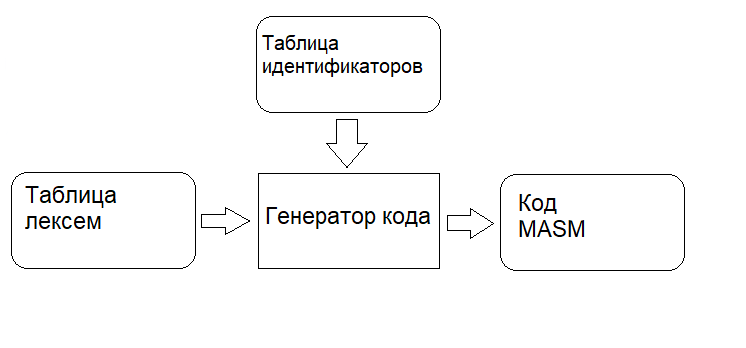


Рисунок 7.1 – Структура генератора кода

Генератор кода последовательно проходит таблицу лексем, при необходимости обращаясь к таблице идентификаторов. В зависимости от пройденных лексем выполняется генерация кода ассемблера.

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

Элементы таблицы идентификаторов расположены в разных сегментах языка ассемблера – .data и .const. Идентификаторы языка MMA-2021 размещены в сегменте данных(.data). Литералы – в сегменте констант (.const). Соответствия между типами данных идентификаторов на языке MMA-2021 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка MMA-2021 и языка Ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке MMA-2021 | Тип идентификатора на языке ассемблера | Пояснение |
| uint | DWORD | Хранит целочисленный беззнаковый тип данных. |
| string | DWORD | Хранит указатель на начало строки. |

**7.3 Статическая библиотека**

В языке MMA-2021 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++. Объявление функций статической библиотеки генерируется автоматически в коде ассемблера.

Стандартная библиотека находится в директории языка и при генерации кода подключается автоматически. Путь к библиотеке генерируется автоматически на стадии генерации кода.

Функции статической библиотеки приведены в таблице 7.3.

Таблица 7.3 – Функции статической библиотеки

|  |  |
| --- | --- |
| Функция | Назначение |
| void outStr(char\* str) | Вывод на консоль строки str |
| void outnum(unsigned long num) | Вывод на консоль целочисленной беззнаковой переменной num |
| unsigned long strLen(char \*str) | Возвращает длину строки sty |
| unsigned long strCmp(char \*str1, \*str2) | Возвращает результат лексикографического сравнения строк str1, str2 |

**7.4 Особенности алгоритма генерации кода**

В языке MMA-2021 генерация кода строится на основе таблиц лексем и идентификаторов. Общая схема работы генератора кода представлена на рисунке 7.2

**7.5 Входные параметры генератора кода**

На вход генератору кода поступают таблицы лексем и идентификаторов исходного код программы на языке MMA-2021 . Результаты работы генератора кода выводятся в файл с расширением .asm.

**7.6 Контрольный пример**

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

# **8 Тестирование транслятора**

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

В языке MMA-2021 не разрешается использовать запрещённые входным алфавитом символы. Результат использования запрещённого символа показан в таблице 8.1.

Таблица 8.1 – Тестирование фазы проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  {  @  } | Ошибка 111: Недопустимый символ в исходном файле (-in), строка 3 ,позиция 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  {  7u  } | Ошибка 122: [Лексическая] Нераспознанная лексема, строка 3 ,слово 7u |
| main  {  x = 5;  return 0;  } | Ошибка 131: [Лексическая] Необъявленный идентификатор, строка 3 |
| string function fi(uint x)  {  return ‘a’;  } | Ошибка 129: [Лексическая] Отсутствие входной точки программы функции main |
| main  {  return 0;  }  main  {  return 1;  } | Ошибка 128: [Лексическая] Повторное объявление main |

Продолжение таблицы 8.3

|  |  |
| --- | --- |
| main  {  new uint abcdefghjkl;  return 0;  } | Ошибка 133: [Лексическая] Превышена длина идентификатора, строка 3 |

На этапе лексического анализа обрабатываются ошибки, которые препятствую правильному построению таблицы лексем и идентификаторов.

## **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 |
| uint function fi(uint p, 6) | 604: строка 1,[Синтаксическая] Ошибка в списке параметров функции  604: строка 1,[Синтаксическая] Ошибка в списке параметров функции  601: строка 1,[Синтаксическая] Не найден список параметров функции  Ошибка 613: [Синтаксическая] Синтаксический анализ завершён с ошибкой |
| main  {  output 5;  } | 612: строка 4,[Синтаксическая] Недопустимая синтаксическая конструкция  610: строка 3,[Синтаксическая] Неверное выражение. Ожидаются только идентификаторы/литералы  610: строка 3,[Синтаксическая] Неверное выражение. Ожидаются только идентификаторы/литералы  Ошибка 613: [Синтаксическая] Синтаксический анализ завершён с ошибкой |
| main  {  if (5 + 7)  {  output 10;  }  } | 608: строка 3,[Синтаксическая] Неверный условный оператор  608: строка 3,[Синтаксическая] Неверный условный оператор  608: строка 3,[Синтаксическая] Неверный условный оператор  Ошибка 613: [Синтаксическая] Синтаксический анализ завершён с ошибкой |

## **8.4 Тестирование семантического анализатора**

Итоги тестирования семантического анализатора приведены в таблице 8.4.

Таблица 8.4 – Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  {  new uint x = 5 + 'z';  return 0;  } | Ошибка 308: [Cемантическая] Недопустимая операция над данным типом, строка 3 |
| main  {  return 'z';  } | Ошибка 300: [Cемантическая] Функция main должна возвращать числовое значение, строка 3 |

## **Заключение**

В ходе выполнения курсовой работы был разработан транслятор для языка программирования MMA-2021. Таким образом, были выполнены основные задачи данной курсовой работы:

* Сформулирована спецификация языка MMA-2021;
* Разработаны конечные автоматы и алгоритмы для реализация лексического анализатора;
* Разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
* Разработан семантический анализатор, осуществляющий проверку смысла используемых инструкций;
* Разработан транслятор с языка программирования MMA-2021 на язык низкого уровня Assembler;
* Проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка MMA-2021 включает:

1. 2 типа данных;
2. Поддержка операции вывода;
3. 2 библиотечные функции
4. Возможность вызова функций стандартной библиотеки;
5. Наличие 5 арифметических операторов для вычисления выражений;
6. Структурированная система для обработки ошибок пользователя.
7. Условный оператор;
8. 6 операторов сравнения для целочисленного типа.

# **Список использованных источников**

1. Ахо, А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003. – 768с.
2. Молчанов, А. Ю. Системное программное обеспечение / А. Ю. Молчанов. – СПб.: Питер, 2010. – 400 с.
3. Ахо, А. Теория синтаксического анализа, перевода и компиляции /А. Ахо, Дж. Ульман. – Москва : Мир, 1998. – Т. 2 : Компиляция. - 487 с.
4. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. - 3-е изд. – Москва : Вильямс, 2003. - 429 с.
5. Орлов, С.А. Теория и практика языков программирования / С.А. Орлов – 2014. – 689 с.
6. Страуструп, Б. Принципы и практика использования C++ / Б. Страуструп – 2009 – 1238 с.

## **Приложение А**

|  |
| --- |
| uint function factorial(uint x)  {  new uint ret = 1;  if (x < 0)  {  ret = 0;  }  if (x == 0)  {  ret = 1;  }  if (x > 0)  {  new uint y = x - 1;  ret = factorial(y) \* x;  }  return ret;  }  new uint function strCmp(string a, string b);  new uint function strLen(string a);  string function strCmpTest()  {  new uint res = strCmp('aaa', 'bbb');  new string outStr;  if (res == 1)  {  outStr = 'Строка 1 больше строки 2';  }  if (res == 2)  {  outStr = 'Строка 2 больше строки 1';  }  if (res == 3)  {  outStr = 'Длина строк не совпадает, или передана пустая строка';  }  return outStr;  }  main  {  new uint x = strLen('hello');  new uint y = factorial(x);  output y;  new string z = strCmpTest();  output z;  return 0;  } |

Листинг 1 - Исходный код на языке MMA-2021

|  |
| --- |
| tfi<0>(ti<1>){dti<2>=l<3>;  e(i<1><l<4>){i<2>=l<4>;  }e(i<1>~l<4>){i<2>=l<3>;  }e(i<1>>l<4>){dti<5>=i<1>-l<3>;  i<2>=i<0>(i<5>)\*i<1>;  }ri<2>;  }dtfi<6>(ti<7>,ti<8>);  dtfi<9>(ti<10>);  tfi<11>(){dti<12>=i<6>(l<13>,l<14>);  dti<15>;  e(i<12>~l<3>){i<15>=l<16>;  }e(i<12>~l<17>){i<15>=l<18>;  }e(i<12>~l<19>){i<15>=l<20>;  }ri<15>;  }m{dti<22>=i<9>(l<23>);  dti<24>=i<0>(i<22>);  pi<24>;  dti<25>=i<11>();  pi<25>;  rl<4>; |

Листинг 2 - Таблица лексем на выходе лексического анализатора

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 3 - Таблица идентификаторов на выходе лексического анализатора

|  |
| --- |
| #define REG\_DECLARE "new"  #define REG\_STRING "string"  #define REG\_INTEGER "uint"  #define REG\_BOOL "bool"  #define REG\_FUNCTION "function"  #define REG\_RETURN "return"  #define REG\_PRINT "output"  #define REG\_MAIN "main"  #define REG\_ID "([a-z]|[A-Z])+"  #define REG\_INTEGER\_LIT "([1-9]+[0-9]\*)|0o?|([1-7]+[0-7]\*o)"  #define REG\_STRING\_LIT "'(.)\*'"  #define REG\_BOOL\_LIT "(true)|(false)"  #define REG\_SEMICOLON ";"  #define REG\_EQUAL "="  #define REG\_COMMA ","  #define REG\_MREQUAL "\\{"  #define REG\_LSEQUAL "\\}"  #define REG\_LEFTTHESIS "\\("  #define REG\_RIGHTTHESIS "\\)"  #define REG\_PLUS "\\+"  #define REG\_MINUS "\\-"  #define REG\_STAR "\\\*"  #define REG\_DIRSLASH "/"  #define REG\_REM\_AFTER\_DIVIDING "%"  #define REG\_IF "if"  #define REG\_ELSE "else"  #define REG\_MORE ">"  #define REG\_LESS "<"  #define REG\_EXCLAMATION "!" |

Листинг 4 - Регулярные выражения для лексического распознавателя

## **Приложение Б**

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1 – Грамматика языка MMA-2021 (продолжение)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2 – Грамматика языка MMA-2021 (продолжение)

## **Приложение В**

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1 – Структура магазинного автомата

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2 – Структура магазинного автомата (продолжение)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3 – Структура магазинного автомата (продолжение)

## **Приложение Г**

|  |
| --- |
| Шаг : Правило Входная лента Стек  0 : S->tfiPTS S$  0 : SAVESTATE: 1  0 : tfiPTS$  1 : fi(ti){dti=l;e(i<l){i=l;} fiPTS$  2 : i(ti){dti=l;e(i<l){i=l;}e iPTS$  3 : (ti){dti=l;e(i<l){i=l;}e( PTS$  4 : P->(E) (ti){dti=l;e(i<l){i=l;}e( PTS$  4 : SAVESTATE: 2  4 : (ti){dti=l;e(i<l){i=l;}e( (E)TS$  5 : ti){dti=l;e(i<l){i=l;}e(i E)TS$  6 : E->ti,E ti){dti=l;e(i<l){i=l;}e(i E)TS$  6 : SAVESTATE: 3  6 : ti){dti=l;e(i<l){i=l;}e(i ti,E)TS$  7 : i){dti=l;e(i<l){i=l;}e(i~ i,E)TS$  8 : ){dti=l;e(i<l){i=l;}e(i~l ,E)TS$  9 : 2  9 : RESSTATE  9 : ti){dti=l;e(i<l){i=l;}e(i E)TS$  10 : E->ti ti){dti=l;e(i<l){i=l;}e(i E)TS$  10 : SAVESTATE: 3  10 : ti){dti=l;e(i<l){i=l;}e(i ti)TS$  11 : i){dti=l;e(i<l){i=l;}e(i~ i)TS$  12 : ){dti=l;e(i<l){i=l;}e(i~l )TS$  13 : {dti=l;e(i<l){i=l;}e(i~l) TS$  14 : T->{rV;} {dti=l;e(i<l){i=l;}e(i~l) TS$  14 : SAVESTATE: 4  14 : {dti=l;e(i<l){i=l;}e(i~l) {rV;}S$  15 : dti=l;e(i<l){i=l;}e(i~l){ rV;}S$  16 : 2  16 : RESSTATE  16 : {dti=l;e(i<l){i=l;}e(i~l) TS$  17 : T->{KrV;} {dti=l;e(i<l){i=l;}e(i~l) TS$  17 : SAVESTATE: 4  17 : {dti=l;e(i<l){i=l;}e(i~l) {KrV;}S$  18 : dti=l;e(i<l){i=l;}e(i~l){ KrV;}S$  19 : K->dti=W;K dti=l;e(i<l){i=l;}e(i~l){ KrV;}S$  19 : SAVESTATE: 5  19 : dti=l;e(i<l){i=l;}e(i~l){ dti=W;KrV;}S$  20 : ti=l;e(i<l){i=l;}e(i~l){i ti=W;KrV;}S$  21 : i=l;e(i<l){i=l;}e(i~l){i= i=W;KrV;}S$  22 : =l;e(i<l){i=l;}e(i~l){i=l =W;KrV;}S$  23 : l;e(i<l){i=l;}e(i~l){i=l; W;KrV;}S$  24 : W->l l;e(i<l){i=l;}e(i~l){i=l; W;KrV;}S$ |

Листинг 1 – Начало разбора синтаксического анализатора

|  |
| --- |
| 989 : W->iF i();pi;rl;}}}}}}}}}}}}}}} W;KrV;}$  989 : SAVESTATE: 76  989 : i();pi;rl;}}}}}}}}}}}}}}} iF;KrV;}$  990 : ();pi;rl;}}}}}}}}}}}}}}}} F;KrV;}$  991 : F->(N) ();pi;rl;}}}}}}}}}}}}}}}} F;KrV;}$  991 : SAVESTATE: 77  991 : ();pi;rl;}}}}}}}}}}}}}}}} (N);KrV;}$  992 : );pi;rl;}}}}}}}}}}}}}}}}} N);KrV;}$  993 : TNS\_NORULECHAIN/NS\_NORULE  993 : RESSTATE  993 : ();pi;rl;}}}}}}}}}}}}}}}} F;KrV;}$  994 : F->() ();pi;rl;}}}}}}}}}}}}}}}} F;KrV;}$  994 : SAVESTATE: 77  994 : ();pi;rl;}}}}}}}}}}}}}}}} ();KrV;}$  995 : );pi;rl;}}}}}}}}}}}}}}}}} );KrV;}$  996 : ;pi;rl;}}}}}}}}}}}}}}}}}} ;KrV;}$  997 : pi;rl;}}}}}}}}}}}}}}}}}}} KrV;}$  998 : K->pV;K pi;rl;}}}}}}}}}}}}}}}}}}} KrV;}$  998 : SAVESTATE: 78  998 : pi;rl;}}}}}}}}}}}}}}}}}}} pV;KrV;}$  999 : i;rl;}}}}}}}}}}}}}}}}}}}} V;KrV;}$  1000: V->i i;rl;}}}}}}}}}}}}}}}}}}}} V;KrV;}$  1000: SAVESTATE: 79  1000: i;rl;}}}}}}}}}}}}}}}}}}}} i;KrV;}$  1001: ;rl;}}}}}}}}}}}}}}}}}}}}} ;KrV;}$  1002: rl;}}}}}}}}}}}}}}}}}}}}}} KrV;}$  1003: TNS\_NORULECHAIN/NS\_NORULE  1003: RESSTATE  1003: i;rl;}}}}}}}}}}}}}}}}}}}} V;KrV;}$  1004: TNS\_NORULECHAIN/NS\_NORULE  1004: RESSTATE  1004: pi;rl;}}}}}}}}}}}}}}}}}}} KrV;}$  1005: K->pV; pi;rl;}}}}}}}}}}}}}}}}}}} KrV;}$  1005: SAVESTATE: 78  1005: pi;rl;}}}}}}}}}}}}}}}}}}} pV;rV;}$  1006: i;rl;}}}}}}}}}}}}}}}}}}}} V;rV;}$  1007: V->i i;rl;}}}}}}}}}}}}}}}}}}}} V;rV;}$  1007: SAVESTATE: 79  1007: i;rl;}}}}}}}}}}}}}}}}}}}} i;rV;}$  1008: ;rl;}}}}}}}}}}}}}}}}}}}}} ;rV;}$  1009: rl;}}}}}}}}}}}}}}}}}}}}}} rV;}$  1010: l;}}}}}}}}}}}}}}}}}}}}}}} V;}$  1011: V->l l;}}}}}}}}}}}}}}}}}}}}}}} V;}$  1011: SAVESTATE: 80  1011: l;}}}}}}}}}}}}}}}}}}}}}}} l;}$  1012: ;}}}}}}}}}}}}}}}}}}}}}}}} ;}$  1013: }}}}}}}}}}}}}}}}}}}}}}}}} }$  1014: }}}}}}}}}}}}}}}}}}}}}}}}} $  1015: 6  1016: ------>LENTA\_END |

Листинг 2 – Конец разбора синтаксического анализатора

|  |
| --- |
| 0 : S->tfiPTS  3 : P->(E)  4 : E->ti  7 : T->{KrV;}  8 : K->dti=W;K  12 : W->l  14 : K->eZ{K}K  15 : Z->(iLl)  17 : L-><  21 : K->i=W;  23 : W->l  26 : K->eZ{K}K  27 : Z->(iLl)  29 : L->~  33 : K->i=W;  35 : W->l  38 : K->eZ{K}  39 : Z->(iLl)  41 : L->>  45 : K->dti=W;K  49 : W->iAW  50 : A->-  51 : W->l  53 : K->i=W;  55 : W->iFAW  56 : F->(N)  57 : N->i  59 : A->\*  60 : W->i  64 : V->i  67 : S->dtfiP;S  71 : P->(E)  72 : E->ti,E  75 : E->ti  79 : S->dtfiP;S  83 : P->(E)  84 : E->ti  88 : S->tfiPTS  91 : P->()  93 : T->{KrV;}  94 : K->dti=W;K  98 : W->iF  99 : F->(N)  100 : N->l,N  102 : N->l  105 : K->dti;K  109 : K->eZ{K}K  110 : Z->(iLl)  112 : L->~  116 : K->i=W;  118 : W->l  121 : K->eZ{K}K  122 : Z->(iLl)  124 : L->~  128 : K->i=W;  130 : W->l  133 : K->eZ{K}  134 : Z->(iLl)  136 : L->~  140 : K->i=W;  142 : W->l  146 : V->i  149 : S->mT  150 : T->{KrV;}  151 : K->dti=W;K  155 : W->iF  156 : F->(N)  157 : N->l  160 : K->dti=W;K  164 : W->iF  165 : F->(N)  166 : N->i  169 : K->pV;K  170 : V->i  172 : K->dti=W;K  176 : W->iF  177 : F->()  180 : K->pV;  181 : V->i  184 : V->l |

Листинг 3 – Пример разбора синтаксического анализатора

## **Приложение Д**

Изображение выглядит как текст

Автоматически созданное описаниеРисунок 1 - Реализация польской нотации

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2 - Реализация польской нотации (продолжение)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3 - Реализация польской нотации (продолжение)

|  |
| --- |
| tfi<0>(ti<1>){dti<2>=l<3>;  e(i<1><l<4>){i<2>=l<4>;  }e(i<1>~l<4>){i<2>=l<3>;  }e(i<1>>l<4>){dti<5>=i<1>l<3>-;  i<2>=i<5>@<0>1i<1>\*;  }ri<2>;  }dtfi<6>(ti<7>,ti<8>);  dtfi<9>(ti<10>);  tfi<11>(){dti<12>=l<13>l<14>@<6>2;  dti<15>;  e(i<12>~l<3>){i<15>=l<16>;  }e(i<12>~l<17>){i<15>=l<18>;  }e(i<12>~l<19>){i<15>=l<20>;  }ri<15>;  }m{dti<22>=l<23>@<9>1;  dti<24>=i<22>@<0>1;  pi<24>;  dti<25>=@<11>0;  pi<25>;  rl<4>;  } |

Листинг 4 - Таблица лексем после преобразования к польской нотации

## **Приложение Е**

Сгенерированный код

|  |
| --- |
| .586  .model flat, stdcall  includelib libucrt.lib  includelib kernel32.lib  includelib ../Debug/MMA\_Lib.lib  ExitProcess PROTO : DWORD  \_strCmp PROTO : DWORD, :DWORD  \_strLen PROTO : DWORD  \_outStr PROTO : DWORD  \_outBool PROTO : DWORD  \_outInt PROTO : DWORD  .stack 4096  .const  \_DIVISION\_BY\_ZERO\_ERROR BYTE 'Ошибка выполнения: деление на ноль', 0  \_OVERFLOW\_ERROR BYTE 'Ошибка выполнения: переполнение', 0  \_NEGATIVE\_RESULT\_ERROR BYTE 'Ошибка выполнения: попытка присвоения отрицательного значения', 0  L0 DWORD 1  L1 DWORD 0  L2 BYTE 'aaa', 0  L3 BYTE 'bbb', 0  L4 BYTE 'Строка 1 больше строки 2', 0  L5 DWORD 2  L6 BYTE 'Строка 2 больше строки 1', 0  L7 DWORD 3  L8 BYTE 'Длина строк не совпадает,или передана пустая строка', 0  L9 BYTE 'hello', 0  .data  \_factorialret DWORD 0 ; uint  \_scope\_2y DWORD 0 ; uint  \_strCmpTestres DWORD 0 ; uint  \_strCmpTestoutStr DWORD ? ; str  \_mainx DWORD 0 ; uint  \_mainy DWORD 0 ; uint  \_mainz DWORD ? ; str  .code  \_factorial PROC \_factorialx: DWORD  push L0  pop \_factorialret  cmp \_factorialret, 0  jl NEGATIVE\_RESULT  push \_factorialx  push L1  pop ebx  pop eax  cmp eax, ebx  jnb FALSE4  push L1  pop \_factorialret  cmp \_factorialret, 0  jl NEGATIVE\_RESULT  FALSE4:  push \_factorialx  push L1  pop ebx  pop eax  cmp eax, ebx  jne FALSE8  push L0  pop \_factorialret  cmp \_factorialret, 0  jl NEGATIVE\_RESULT  FALSE8:  push \_factorialx  push L1  pop ebx  pop eax  cmp eax, ebx  jna FALSE12  push \_factorialx  push L0  pop ebx  pop eax  sub eax, ebx  jo EXIT\_OVERFLOW  push eax  pop \_scope\_2y  cmp \_scope\_2y, 0  jl NEGATIVE\_RESULT  push \_scope\_2y  call \_factorial  push eax  push \_factorialx  pop ebx  pop eax  imul eax, ebx  jo EXIT\_OVERFLOW  push eax  pop \_factorialret  cmp \_factorialret, 0  jl NEGATIVE\_RESULT  FALSE12:  jmp EXIT  EXIT\_DIV\_ON\_NULL:  push offset \_DIVISION\_BY\_ZERO\_ERROR  call \_outStr  push -1  call ExitProcess  EXIT\_OVERFLOW:  push offset \_OVERFLOW\_ERROR  call \_outStr  push -2  call ExitProcess  NEGATIVE\_RESULT:  push offset \_NEGATIVE\_RESULT\_ERROR  call \_outStr  push -3  call ExitProcess  EXIT:  mov eax, \_factorialret  ret 4  \_factorial ENDP  \_strCmpTest PROC  push offset L2  push offset L3  call \_strCmp  push eax  pop \_strCmpTestres  cmp \_strCmpTestres, 0  jl NEGATIVE\_RESULT  push \_strCmpTestres  push L0  pop ebx  pop eax  cmp eax, ebx  jne FALSE25  push offset L4  pop \_strCmpTestoutStr  FALSE25:  push \_strCmpTestres  push L5  pop ebx  pop eax  cmp eax, ebx  jne FALSE29  push offset L6  pop \_strCmpTestoutStr  FALSE29:  push \_strCmpTestres  push L7  pop ebx  pop eax  cmp eax, ebx  jne FALSE33  push offset L8  pop \_strCmpTestoutStr  FALSE33:  jmp EXIT  EXIT\_DIV\_ON\_NULL:  push offset \_DIVISION\_BY\_ZERO\_ERROR  call \_outStr  push -1  call ExitProcess  EXIT\_OVERFLOW:  push offset \_OVERFLOW\_ERROR  call \_outStr  push -2  call ExitProcess  NEGATIVE\_RESULT:  push offset \_NEGATIVE\_RESULT\_ERROR  call \_outStr  push -3  call ExitProcess  EXIT:  mov eax, \_strCmpTestoutStr  ret 0  \_strCmpTest ENDP  main PROC  push offset L9  call \_strLen  push eax  pop \_mainx  cmp \_mainx, 0  jl NEGATIVE\_RESULT  push \_mainx  call \_factorial  push eax  pop \_mainy  cmp \_mainy, 0  jl NEGATIVE\_RESULT  push \_mainy  call \_outInt  call \_strCmpTest  push eax  pop \_mainz  push \_mainz  call \_outStr  jmp EXIT  EXIT\_DIV\_ON\_NULL:  push offset \_DIVISION\_BY\_ZERO\_ERROR  call \_outStr  push -1  call ExitProcess  EXIT\_OVERFLOW:  push offset \_OVERFLOW\_ERROR  call \_outStr  push -2  call ExitProcess  NEGATIVE\_RESULT:  push offset \_NEGATIVE\_RESULT\_ERROR  call \_outStr  push -3  call ExitProcess  EXIT:  push L1  call ExitProcess  main ENDP  end main |

Результат работы

