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 Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора IMS-2020»

Выполнил студент Ивашков Макар Сергеевич

(Ф.И.О.)

Руководитель проекта ст.пр. Наркевич Аделина Сергеевна

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

Заведующий кафедрой к.т.н., доц. Пацей Н.В.

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

Консультанты ст.пр. Наркевич Аделина Сергеевна

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

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

Нормоконтролер ст.пр. Наркевич Аделина Сергеевна

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

Курсовой проект защищен с оценкой

Минск 2020

**Содержание**

[**Введение** 5](#_Toc59058414)

[**Глава 1. Спецификация языка программирования** 6](#_Toc59058415)

[**1.1.** **Характеристика языка программирования** 6](#_Toc59058416)

[**1.2.** **Алфавит языка** 6](#_Toc59058417)

[**1.3.** **Символы сепараторы** 6](#_Toc59058418)

[**1.4.** **Применяемые кодировки** 7](#_Toc59058419)

[**1.5.** **Типы данных** 7](#_Toc59058420)

[**1.6.** **Преобразование типов данных** 8](#_Toc59058421)

[**1.7.** **Идентификаторы** 8](#_Toc59058422)

[**1.8.** **Литералы** 8](#_Toc59058423)

[**1.9.** **Объявление данных** 9](#_Toc59058424)

[**1.10.** **Инициализация данных** 9](#_Toc59058425)

[**1.11.** **Инструкции языка** 10](#_Toc59058426)

[**1.13.** **Выражения и их вычисления** 11](#_Toc59058427)

[**1.14.** **Программные конструкции языка** 11](#_Toc59058428)

[**1.15.** **Область видимости** 12](#_Toc59058429)

[**1.16.** **Семантические проверки** 12](#_Toc59058430)

[**1.17.** **Распределение оперативной памяти на этапе выполнения** 13](#_Toc59058431)

[**1.18.** **Стандартная библиотека и её состав** 13](#_Toc59058432)

[**1.19.** **Ввод и вывод данных** 14](#_Toc59058433)

[**1.20.** **Точка входа** 14](#_Toc59058434)

[**1.21.** **Препроцессор** 14](#_Toc59058435)

[**1.22.** **Соглашения о вызовах** 14](#_Toc59058436)

[**1.23.** **Объектный код** 14](#_Toc59058437)

[**1.24.** **Классификация сообщений транслятора** 14](#_Toc59058438)

[**1.25.** **Контрольный пример** 15](#_Toc59058439)

[**Глава 2. Структура транслятора** 16](#_Toc59058440)

[**2.1 Компоненты транслятора, их назначение и принципы взаимодействия** 16](#_Toc59058441)

[**2.2 Перечень входных параметров транслятора** 17](#_Toc59058442)

[**2.3 Перечень протоколов, формируемых транслятором и их содержимое** 18](#_Toc59058443)

[**Глава 3. Разработка лексического анализатора** 19](#_Toc59058444)

[**3.1 Структура лексического анализатора** 19](#_Toc59058445)

[**3.2 Контроль входных символов** 20](#_Toc59058446)

[**3.3 Удаление избыточных символов** 21](#_Toc59058447)

[**3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов** 21](#_Toc59058448)

[**3.5 Основные структуры данных** 23](#_Toc59058449)

[**3.6 Принцип обработки ошибок** 24](#_Toc59058450)

[**3.7 Структура и перечень сообщений лексического анализатора** 25](#_Toc59058451)

[**3.8 Параметры лексического анализатора и режимы его работы** 25](#_Toc59058452)

[**3.9 Алгоритм лексического анализа** 25](#_Toc59058453)

[**3.10 Контрольный пример** 26](#_Toc59058454)

[**Глава 4. Разработка синтаксического анализатора** 26](#_Toc59058455)

[**4.1 Структура синтаксического анализатора** 26](#_Toc59058456)

[**4.2 Контекстно свободная грамматика, описывающая синтаксис языка** 27](#_Toc59058457)

[**4.3 Построение конечного магазинного автомата** 29](#_Toc59058458)

[**4.4 Основные структуры данных** 30](#_Toc59058459)

[**4.5 Описание алгоритма синтаксического разбора** 30](#_Toc59058460)

[**4.6 Структура и перечень сообщений синтаксического анализатора** 31](#_Toc59058461)

[**4.7 Параметры синтаксического анализатора и режимы его работы** 31](#_Toc59058462)

[**4.8 Принцип обработки ошибок** 31](#_Toc59058463)

[**4.9 Контрольный пример** 31](#_Toc59058464)

[**Глава 5. Разработка семантического анализатора** 31](#_Toc59058465)

[**5.1 Структура семантического анализатора** 31](#_Toc59058466)

[**5.2 Функции семантического анализатора** 32](#_Toc59058467)

[**5.3 Структура и перечень сообщений семантического анализатора** 32](#_Toc59058468)

[**5.4 Принцип обработки ошибок** 32](#_Toc59058469)

[**5.5 Контрольный пример** 33](#_Toc59058470)

[**Глава 6. Вычисление выражений** 34](#_Toc59058471)

[**6.1 Выражения, допускаемые языком** 34](#_Toc59058472)

[**6.2 Польская запись** 34](#_Toc59058473)

[**6.3 Программная реализация обработки выражений** 35](#_Toc59058474)

[**6.4 Контрольный пример** 35](#_Toc59058475)

[**Глава 7. Генерация кода** 36](#_Toc59058476)

[**7.1 Структура генератора кода** 36](#_Toc59058477)

[**7.2 Представление типов данных в оперативной памяти** 36](#_Toc59058478)

[**7.3 Статическая библиотека** 37](#_Toc59058479)

[**7.4.** **Особенности генерации кода** 37](#_Toc59058480)

[**7.5 Входные параметры генератора кода** 38](#_Toc59058481)

[**7.6 Контрольный пример** 38](#_Toc59058482)

[**Глава 8. Тестирование транслятора** 38](#_Toc59058483)

[**8.1 Тестирование фазы проверки на допустимость символов** 38](#_Toc59058484)

[**8.2** **Тестирование лексического анализатора** 39](#_Toc59058485)

[**8.3 Тестирование синтаксического анализатора** 39](#_Toc59058486)

[**8.4 Тестирование семантического анализатора** 40](#_Toc59058487)

[**Заключение** 41](#_Toc59058488)

[**Приложения** 42](#_Toc59058489)

[**Контрольный пример** 42](#_Toc59058490)

[**Приложение А** 43](#_Toc59058491)

[**Приложение Б** 50](#_Toc59058492)

[**Приложение В** 51](#_Toc59058493)

[**Приложение Г** 53](#_Toc59058494)

[**Приложение Д** 54](#_Toc59058495)

[**Приложение Е** 57](#_Toc59058496)

[**Литература** 58](#_Toc59058497)

# **Введение**

Целью курсового проекта поставлена задача разработки компилятора для моего языка программирования IMS-2020. Этот язык программирования предназначен для выполнения простейших операций и арифметических действий над числами и строками.

Компилятор IMS-2020 – это программа, задачей которого является перевод программы, написанной на языке программирования IMS-2020 в программу на язык ассемблера.

Транслятор IMS-2020 состоит из следующих частей:

– лексический и семантический анализаторы;

– синтаксический 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 буду приведены в соответствующих главах курсового проекта.

# **Глава 1. Спецификация языка программирования**

## **Характеристика языка программирования**

Язык программирования IMS-2020 классифицируется как процедурный, универсальный, строготипизированный, компилируемый и не объектно-ориентированный язык.

* 1. **Алфавит языка**

В языке IMS-2020 могут использоваться символы латинского алфавита, цифры десятичной системы счисления от 0 до 9, спецсимволы, а также непечатные символы пробела, табуляции и перевода строки. Русские символы разрешены только в строковых литералах.

## **Символы сепараторы**

Символы, которые являются сепараторами представлены в таблице 1.1.

Таблица 1.1 – Сепараторы

|  |  |
| --- | --- |
| **Символ-сепаратор** | **Описание** |
| пробел | Допускается везде кроме идентификаторов и ключевых слов; |
| ; | Разделитель инструкций |
| {…} | Программный блок |
| (…) | Параметры |
| (…) | Приоритетность операций |
| […] | Доступ к элементам массива |
| $...$ | Указание родительского блока |
| as | Оператор присваивания |
| , | Разделение параметров |
| + - \* % | Арифметические операции |
| > < \ / = | Логические операции |
| ‘…’ | Обозначение строкового литерала |

## **Применяемые кодировки**

Для написания исходного кода на языке программирования IMS-2020 используется кодировка Windows-1251, представленная на рисунке 1.1



Рисунок 1.1 – Применяемая кодировка символов

## **Типы данных**

В языке IMS-2020 реализованы два типа данных: целочисленный и строковый. Описание типов данных, предусмотренных в данным языке представлено в таблице 1.2.

Таблица 1.2 – Типы данных языка IMS-2020

|  |  |
| --- | --- |
| **Тип данных** | **Описание** |
| integer | Беззнаковые целочисленные данные (четыре байта, диапазон от 0 до 231 -1), автоматическая инициализация 0; |
| string | Строка, любые символы, (максимально 255 символов), автоматическая инициализация пустой строкой длины 0; |
| logic | Логический тип данных (4 байта, true или false), автоматическая инициализация false; |
| integer [<литерал>] | Целочисленный массив, каждый элемент размером 4 байта, по умолчанию каждый элемент инициализируется нулём; переменную такого типа можно инициализировать некоторым набором целочисленных литералов внутри блока declaration {…}; |

## **Преобразование типов данных**

Преобразование типов данных не поддерживается, т.е. язык является строготипизированным.

## **Идентификаторы**

В имени идентификатора допускаются только символы латинского алфавита нижнего и верхнего регистров. Максимальная длина имени идентификатора - 32 символа. Общее количество идентификаторов ограничено максимальным размером таблицы идентификаторов. При вводе идентификатора длиной более разрешенного количества символов компилятор уведомит пользователя об ошибке. Имя идентификатора не может совпадать с ключевыми словами.

## **Литералы**

С помощью литералов осуществляется инициализация переменных. В языке существует два типа литералов. Краткое описание литералов языка IMS-2020 представлено в таблице 1.3.

Таблица 1.3 – Описание литералов

|  |  |  |  |
| --- | --- | --- | --- |
| Тип литерала | Регулярное выражение | Описание | Пример |
| Целочисленный литерал | [0-9]+[0-9]\* | Целочисленные неотрицательные литералы, по умолчанию инициализируются 0. Литералы могут как восьмеричными, так и десятичными. Восьмеричный литерал всегда начинается с 0. | …  integer a;  integer b;  …  a as 4; - инициалзация переменной десятичным литералом  b as 012; - инициалзация переменной восьмеричным литералом |
| Строковые литерал | [a-z|A-Z|А-Я|а-я|0-9|!-/]+ | Символы, заключённые в ‘’ (одинарные кавычки), являются строковым литералом, по умолчанию инициализируются пустой строкой. | …  string sc;  …  Sc as ‘hello’; - инициализация пременной строковым литералом |
| Логический литерал | [false|true]+ | Ключевые слова true/false являются литералами типа logic | …  logic lt  …  lt as true; - инициализация логической переменной логическим литералом |

## **Объявление данных**

Объявление всех типов данных, включая переменные и функции, в языке программирования IMS-2020 может происходить только внутри специального блока объявления declaration{…}. В случае, если необходимо областью видимости переменной объявить функцию, необходимо после объявления самой функции объявить эту переменную и после указания типа данных заключить между двух знаков ‘$’ имя функции, внутри которой данная переменная будет использоваться.

## **Инициализация данных**

При объявлении переменной не допускается инициализация данных. Инициализация возможна в любом месте программы, кроме блока объявлений declaration{…}. Краткое описание способов инициализации переменных языка IMS-2020 представлено в таблице 1.4.

Таблица 1.4 – Способы инициализации переменных

|  |  |  |
| --- | --- | --- |
| Конструкция | Описание | Пример |
| <тип данных> <идентификатор>; | Автоматическая инициализация: переменные типа integer инициализируются нулём, переменные типа string – пустой строкой. | integer a;  string sc; |
| <идентификатор> as <значение>; | Присваивание переменной значения. | a as 1;  sc as ‘hello’; |

## **Инструкции языка**

Все возможные инструкции языка программирования IMS-2020 представлены в общем виде в таблице 1.5.

Таблица 1.5 – Инструкции языка программирования IMS - 2020

|  |  |
| --- | --- |
| **Инструкция** | **Описание** |
| <тип данных> $индентификатор функции$ идентификатор; | Объявление переменных с областью видимости внутри пользовательской функции; |
| <тип данных> идентификатор; | Объявление переменных с областью видимости внутри функции main |
| <тип данных> function идентификатор (<тип данных> идентификатор,... ); | Объявление внешних пользовательских функций; |
| as | присвоение значения; |
| print <идентификатор или литерал> ; | вывод в стандартный поток вывода |
| if (<идентификатор>/<литерал>< логическая операция> <идентификатор>/ <литерал>) {…} | Условный оператор, в случае истинности выражения в скобках выполняется блок инструкций, заключенный в фигурные скобки |
| while (<идентификатор>/<литерал>< логическая операция> <идентификатор>/ <литерал>) {…} | Оператор цикла, блок инструкций выполняется до тех пор, пока истинно условие в скобках |
| integer [<литерал>] <идентификатор> <литерал>, …. | Объявление массива целочисленных значений |

* 1. **Операции языка**

Язык программирования IMS-2020 может выполнять бинарные операции, представленные в таблице 1.6.

Таблица 1.6 – Операции языка программирования IMS-2020

|  |  |
| --- | --- |
| **Бинарная операция** | **Описание** |
| + | суммирование, (integer, integer); |
| + | конкатенация, (string, string); |
| - | вычитание, (integer, integer); |
| \* | умножение, (integer, integer); |
| % | деление, (integer, integer); |
| = | равенство, (integer, integer); |
| < | меньше, (integer, integer); |
| > | больше, (integer, integer); |
| / | больше либо равно, (integer, integer); |
| \ | меньше либо равно, (integer, integer); |

В качестве

## **Выражения и их вычисления**

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

* Допускается использовать скобки для смены приоритета операций;
* Выражение записывается в строку без переносов;
* Использование двух подряд идущих операторов не допускается;
* 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 ассемблер

## **Программные конструкции языка**

Ключевые программные конструкции языка программирования IMS-2020 представлены в таблице 1.7.

Таблица 1.7 – Программные конструкции языка IMS-2020

|  |  |
| --- | --- |
| **Конструкция** | **Описание** |
| Main  {  return <integer-идентификатор> или <integer-литерал>;  } | Главная функция |
| Declare  {  …  } | Блок объявления переменных |
| <тип данных> function <идентификатор> (<тип данных> <идентификатор>, ...)  {  return <идентификатор> или <литерал> ;  } | Пользовательские функции |

## **Область видимости**

Областью видимости идентификатора по умолчанию является функция main. В случае, если идентификатор принадлежит какой-либо другой функции, имя которой будет заключено между знаками “$” после указания типа данных, область видимости переменной будет только внутри этой функции.

## **Семантические проверки**

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

Таблица 1.8 – Семантические проверки

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Идентификаторы не должны повторяться |
| 2 | Тип данных передаваемых значений в функцию должен совпадать с типом параметров при её объявлении |
| 3 | В функцию должны быть переданы параметры |
| 4 | Тип данных результата выражения должен совпадать с типом данных идентификатора, которому оно присваивается |
| 5 | Индекс вызываемого элемента массива не должен превосходить размер массива |
| 6 | Функция main обязательно должна присутствовать и быть единственной |
| 7 | Блок объявления declaration обязательно должен присутствовать |
| 8 | Параметру функции нельзя присвоить значение |

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

Все переменные размещаются в куче.

## **Стандартная библиотека и её состав**

Стандартная библиотека IMS-2020 написана на языке программирования C++. Функции стандартной библиотеки с описанием представлены в таблице 1.9.

Таблица 1.9 – Состав стандартной библиотеки

|  |  |  |
| --- | --- | --- |
| Функция(C++) | Возвращаемое значение (IMS-2020) | Описание |
| void isvisokosni(int value); | integer | Функция проверяет, является ли число, переданное параметром, високосным годом или нет, и выводит в консоль соответствующее диагностическое сообщение |
| void printcurrentdate(); | integer | Функция выводит в консоль текущую дату и время |
| void print(char\* value); | - | Функция выводит в консоль строковый литерал или переменную, переданный ей через параметры |
| void printint(int value); | - | Функция выводит в консоль целочисленный литерал или переменную, переданный ей через параметры |
| Void concat1(char\* dest, char\* par1) | - | Функция выполняет присваивание строковой переменной строкового значения |
| Void concat2(char\* destination, char\* left\_operand, char\* right\_operand) | - | Функция выполняет конкатенацию двух строковых значений |

## **Ввод и вывод данных**

В языке IMS-2020 не реализованы средства ввода данных.

Для вывода данных в стандартный поток вывода предусмотрена функция print, в зависимости от принимаемого значения которой вызывается она из соответствующих функций стандартной библиотеки описаны в таблице 1.9.

## **Точка входа**

В языке IMS-2020 каждая программа должна содержать главную функцию main, т. е. точку входа, с которой начнется последовательное выполнение программы.

## **Препроцессор**

Препроцессор в языке программирования IMS-2020 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**

IMS-2020 транслируется в язык ассемблера.

## **Классификация сообщений транслятора**

В случае возникновения ошибки в коде программы на языке IMS-2020 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 |
| 120-129 | Ошибки лексического анализа |
| 130-139 | Ошибки синтаксического анализа |
| 140-160 | Ошибки семантического анализа |

## **Контрольный пример**

Контрольный пример представлен в главе Приложения.

# **Глава 2. Структура транслятора**

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

Транслятор преобразует программу, написанную на языке IMS-2020 в программу на языке ассемблера. Для указания выходных файлов используются входные параметры транслятора, которые описаны в пункте 2.2. 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

Первая стадия работы компилятора называется лексическим анализом, а про-грамма, её реализующая, – лексическим анализатором (сканером). На вход лексиче-ского анализатора подаётся последовательность символов входного языка. Он про-изводит предварительный разбор текста, преобразующий единый массив текстовых символов в массив отдельных слов (в теории компиляции вместо термина «слово» часто используют термин «токен»). Примеры лексических единиц: идентифика-торы, числа, символы операций, служебные слова и т.д. Лексический анализатор преобразует исходный текст, заменяя лексические единицы их внутренним пред-ставлением – лексемами, для создания промежуточного представления исходной программы. Каждой лексеме сопоставляется ее тип и запись в таблице идентифика-торов, в которой хранится дополнительная информация.

Таблица лексем (ТЛ) и таблица идентификаторов (ТИ) являются входом для следующей фазы компилятора – синтаксического анализа (разбора, парсера).

Цели лексического анализатора:

− убрать все лишние пробелы;

− выполнить распознавание лексем;

− построить таблицу лексем и таблицу идентификаторов;

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

Семантический анализ в свою очередь является проверкой исходной программы IMS-2020 на семантическую согласованность с определением языка, т.е. проверяет правильность текста исходной программы с точки зрения семантики.

Синтаксический анализ – это основная часть транслятора, предназначенная для распознавания синтаксических конструкций и формирования промежуточного кода IMS-2020. Для этого используются таблица лексем и идентификаторов. Синтаксический анализатор распознаёт синтаксические конструкции, выявляет синтаксические ошибки при их наличии и формирует дерево разбора

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 IMS-2020, прошедший все предыдущие этапы, в код на языке Ассемблера.

## **2.2. Перечень входных параметров транслятора**

Входные параметры представлены в таблице 2.1.

Таблица 2.1 — Входные параметры транслятора языка IMS-2020

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<имя\_файла> | Входной файл с расширением .txt, в котором содержится исходный код на IMS-2020. 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 IMS-2020. В этот файл могут быть выведены таблицы идентификаторов, лексем, а также дерево разбора. | <имя\_файла>.log |

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

Таблица с перечнем протоколов, формируемых транслятором языка IMS-2020 и их назначением представлена в таблице 2.2.

Таблица 2.2 – Протоколы, формируемые транслятором языка IMS-2020

|  |  |
| --- | --- |
| Формируемый протокол | Описание протокола |
| Файл журнала, “log.txt” | Файл содержит в себе краткую информацию об исходном коде на языке IMS-2020. В этот файл могут быть выведены таблицы идентификаторов, лексем, а также сообщения о возникших ошибках. |
| “IMS\_2020asm.asm” | Содержит сгенерированный код на языке Ассемблера. |

# **Глава 3. Разработка лексического анализатора**

## **3.1. Структура лексического анализатора**

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

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

Функции лексического анализатора:

− удаление «пустых» символов и комментариев. Если «пустые» символы (пробелы, знаки табуляции и перехода на новую строку) и комментарии будут уда-лены лексическим анализатором, синтаксический анализатор никогда не столкнется с ними (альтернативный способ, состоящий в модификации грамматики для вклю-чения «пустых» символов и комментариев в синтаксис, достаточно сложен для реа-лизации);

− распознавание идентификаторов и ключевых слов;

− распознавание констант;

− распознавание разделителей и знаков операций.

Исходный код программы представлен в приложении А, структура лексиче-ского анализатора представлена на рисунке 3.1.



Рисунок 3.1 — Структура лексического анализатора IMS-2020

## **3.2.. Контроль входных символов**

Исходный код на языке программирования IMS-2020 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 на рисунке 3.2.

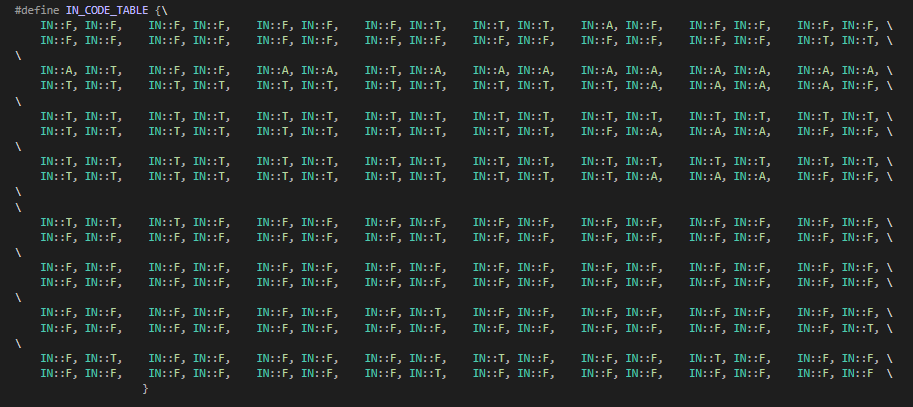


Рисунок 3.2 —Таблица контроля входных символов

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, A – символ – сепаратор, I – игнорируемый символ.

## **3.3. Удаление избыточных символов**

Избыточными символами на языке IMS-2020 могут сичтаться пробел, следующий за другим пробелом, а также символ табуляции.  
Описание алгоритма удаления избыточных символов:

1) Посимвольно считываем файл с исходным кодом программы;

2) Встреча второго подряд 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, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов**

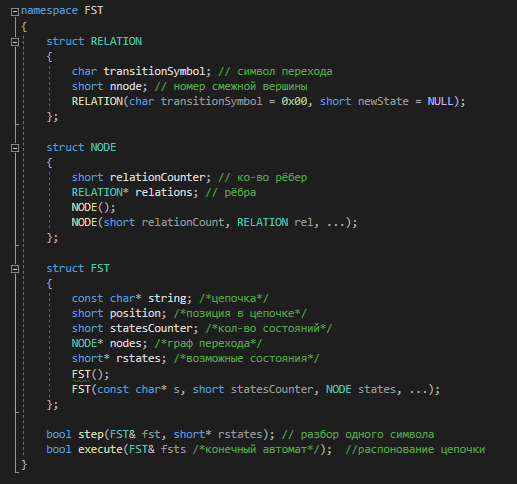
Лексемы – это символы, соответствующие ключевым словам, символам операций и сепараторам, необходимые для упрощения дальнейшей обработки исходного кода программы. Данное соответствие описано в таблице 3.1.

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

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Пояснение |
| main | m | Главная функция. |
| as | a | Объявление переменной. |
| function | f | Объявление функции. |
| print | p | Ввод данных. |
| return | r | Возврат функцией значения |
| while | w | Оператор цикла |
| integer, string, logic | t | Названия типов данных языка. |
| Идентификатор | i | Идентификатор |
| Литерал | l | Литерал любого доступного типа. |
| if | I | Оператор условного блока |
| $ | $ | Указание принадлежности переменной к какой-либо функции |
| ; | ; | Разделение выражений. |
| , | , | Разделение параметров функций. |
| +  -  \*  % | +  -  \*  / | Знаки арифметических операций. |
| >  <  \  /  = | >  <  d  /  = | Знаки логических операторов |
| [ | [ | Начало блока/тела функции. |
| ] | ] | Закрытие блока/тела функции. |
| }  { | }  { | Знаки сдвиговых операций. |

Пример реализации таблицы лексем представлен в приложении А.

Каждому выражению соответствует детерминированный конечный автомат, по которому происходит разбор данного выражения. На каждый автомат в массиве подаётся токен и с помощью регулярного выражения, соответствующего данному графу переходов, происходит разбор. В случае успешного разбора выражения оно записывается в таблицу лексем. Если выражение является идентификатором или литералом, информация также заносится в таблицу идентификаторов. Структура конечного автомата и пример графа перехода конечного автомата изображены на рисунках 3.3 и 3.4 соответственно.



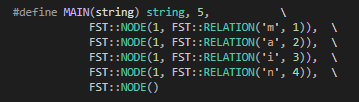
Рисунок 3.3 – Структура конечного автомата

Рисунок 3.4 – Пример реализации графа конечного автомата для токена *main* **(**точки входа)

## **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), номер в таблице лексем (idxfirstLE), тип данных (iddatatype), тип идентификатора (idtype) и значение (или параметры функций) (value). Код C++ со структурой таблицы лексем представлен на рисунке 3.5. Код C++ со структурой таблицы идентификаторов представлен на рисунке 3.6.

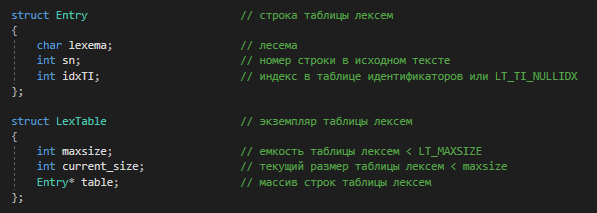


Рисунок 3.5 – Структура таблицы лексем

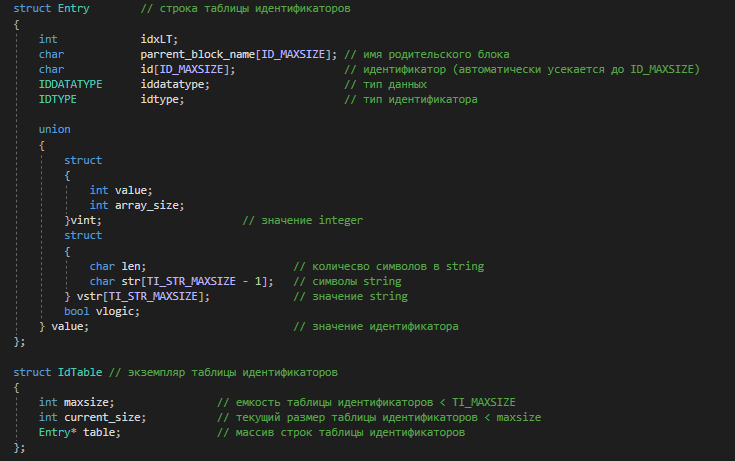


Рисунок 3.6 – Структура таблицы идентификаторов

## **3.6. Принцип обработки ошибок**

При возникновении ошибки типа предупреждение транслятор продолжает свою работу, а предупреждения записываются в специальную структуру с номером ошибки и диагностическим сообщением.

Когда возникает критическая ошибка – работа транслятора прекращается.

В случаях, если на этапе было найдено менее 5 предупреждений, они будут выведены в файл после окончания этапа трансляции.

В случае, если была найдена критическая ошибка или в структуре находятся 5 предупреждений – работа транслятора прекращается и диагностическое сообщение будет выведено в файл.

Перечень сообщений представлен на рисунке 3.7.

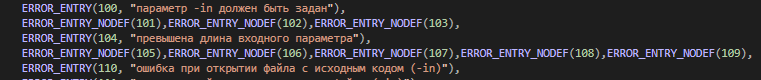


Рисунок 3.7 – Сообщения лексического анализатора

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

Индексы ошибок, обнаруживаемых лексическим анализатором, находятся в диапазоне 120-129.

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

Входные параметры используются для вывода результата работы лексического анализатора. Представлены в таблице 3.2.

Таблица 3.2 Входные параметры транслятора языка IMS-2020

|  |  |
| --- | --- |
| Входной параметр | Описание |
| -lex | Параметр для вывода таблицы лексем в файл протокола |
| -id | Параметр для вывода таблицы идентификаторов в файл протокола |

## **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 автомат не был подобран, запоминается номер строки, в которой находился этот токен и в последствии будет выведено 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 знаком арифметической операции либо функцией стандартной библиотеки, то он заносится в таблицу идентификаторов.

В случае, если является литералом, то заносится в таблицу идентификаторов с именем “literal<n>”, где n является номером литерала.

Когда встречаем токен, являющийся ключевым словом, которое отвечает за тип данных, заносим лексему, соответствующую ему, в таблицу лексем и запоминаем тип данных, которому он соответствует. В последствии, когда встречаем идентификатор, заносим его в таблицу идентификаторов с соответствующим ему типом данных и именем вида “aaaaabbbcc”, где a – символы из имени идентификатора, b – постфикс из имени функции, в которой он находится и c – номер функции.

Пример. Регулярное выражение для ключевого слова main: ‘main’.

Граф конечного автомата для этой лексемы представлен на рисунке 3.8. S0 – начальное состояние, S4 – конечное состояние автомата.

m

i

n

a

Рисунок 3.8 – Граф переходов для цепочки ‘road’

## **3.10. Контрольный пример**

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

# **Глава 4. Разработка синтаксического анализатора**

## **4.1. Структура синтаксического анализатора**

Синтаксический анализ – это фаза трансляции, выполняемая после лексического анализа и предназначенная для распознавания синтаксических конструкций IMS-2020. 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 IMS-2020

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

В синтаксическом анализаторе транслятора языка IMS-2020 используется контекстно-свободная грамматика , где

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

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

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

S – начальный символ грамматики, являющийся нетерминалом.

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

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

Грамматика языка IMS-2020 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 IMS-2020

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| S | tfi(F){NrE;}S  d{N}S  m{NrB;}  tfi(){NrE;}S | Порождает правила, описывающее общую структуру программы |
| N | i(W);N  i(W);  ti;N  ti;  t$i$i;N  t$i$i;  tfi(F);N  tfi();N  tfi(F);  tfi();  t[l]iL;N  t[l]iL;  t$i$ [l]iL;N  t$i$ [l]iL;  iaE;N  iaE;  pB;N  pB;  I(C){N}N  I(C){N}  I(C){rE;}N  I(C){rE;}  I(C){NrE;}N  I(C){NrE;}  w(C){N}N  w(C){N} | Порождает правила, описывающие конструкции языка |
| E | i  l  i[i]  i[l]  i[i]M  i[l]M  (E)  i(W)  i(W)M  iM  lM  (E)M | Порождает правила, описывающие выражения |
| L | l  l, L | Порождает правила, описывающие инициализацию массива |
| B | i  l  i[i]  i[l] | Порождает правила, описывающие операнды логического выражения |
| M | +E  -E  \*E  %E  +EM  -EM  \*EM  %EM | Порождает правила, описывающие арифметические операции |
| C | iOB  lOB | Порождает правила, описывающие строение логической инструкции |
| O | <  >  b  /  = | Порождает правила, описывающие логические операции |
| F | ti  ti, F | Порождает правила, описывающие параметры функций |
| W | i  l  i, W  l, W | Порождает правила, опеределяющие выражения, которые могут быть переданы в функции в качестве параметров |

## **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 |
|  | 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 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 |

## **4.4. Основные структуры данных**

Основные структуры данных синтаксического анализатора включают в себя структуру магазинного конечного автомата и структуру грамматики Грейбах, описывающей правила языка IMS-2020. Данные структуры представлены в приложении В.

## **4.5. Описание алгоритма синтаксического разбора**

Принцип работы автомата следующий:

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

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

Перечень сообщений синтаксического анализатора представлен на рисунке 4.1.

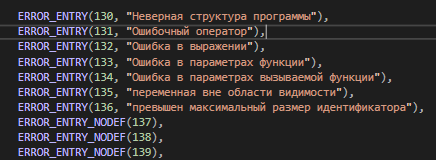


Рисунок 4.1 – Перечень сообщений синтаксического анализатора

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

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

## **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 в протокол будет выведено диагностическое сообщение.

## **4.9. Контрольный пример**

Пример разбора синтаксическим анализатором исходного кода на языке IMS-2020 представлен в приложении Г. Дерево разбора исходного кода также представлено в приложении Г.

# **Глава 5. 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 анализатор

## **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 140. Сообщения, формируемые семантическим анализатором, представлены на рисунке 5.2.

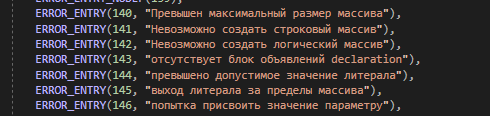


Рисунок 5.2 – Перечень сообщений семантического анализатора

## **5.4. Принцип обработки ошибок**

Ошибки, возникающие в процессе трансляции программы, фиксируются в протокол, заданный входным параметрами. В случае возникновения ошибок происходит их протоколирование с номером ошибки и диагностическим сообщением. Анализ останавливается после того, как будут найдены все ошибки.

## **5.5. Контрольный пример**

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

# **Глава 6. Вычисление выражений**

## **6.1. Выражения, допускаемые языком**

В языке IMS-2020 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 IMS-2020

|  |  |
| --- | --- |
| Операция | Значение приоритета |
| ( ) | 3 |
| \* | 2 |
| % | 2 |
| + | 1 |
| - | 1 |
| } | 0 |
| { | 0 |

## **6.2.. Польская запись**

Выражения в языке IMS-2020 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:

– читаем очередной символ;

– если он является идентификатором или литералом, то добавляем его к выходной строке;

– если символ является символом функции, то помещаем его в стек;

– если символ является открывающей скобкой, то она помещается в стек;

– исходная строка просматривается слева направо;

– если символ является закрывающей скобкой, то выталкиваем из стека в выходную строку все символы пока не встретим открывающую скобку. При этом обе скобки удаляются и не попадают в выходную строку;

– как только входная лента закончится все символы из стека выталкиваются в выходную строку;

– в случае если встречаются операции, то выталкиваем из стека в выходную строку все операции, которые имеют выше приоритетность чем последняя операция;

– также, если идентификатор является именем функции, то он заменяется на спецсимвол «@».

## **6.3. Программная реализация обработки выражений**

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

## **6.4. Контрольный пример**

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

Таблица 6.2. – Преобразование выражений к ПОЛИЗ

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

# **Глава 7. Генерация кода**

## **7.1. Структура генератора кода**

Генерация объектного кода — это перевод компилятором внутреннего представления исходной программы IMS-2020 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

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

Элементы таблицы идентификаторов расположены в разных сегментах языка ассемблера – .data и .const. Идентификаторы языка IMS-2020 размещены в сегменте данных(.data). Литералы – в сегменте констант (.const). Соответствия между типами данных идентификаторов на языке IMS-2020 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка IMS-2020 и языка Ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке IMS-2020 | Тип идентификатора на языке ассемблера | Пояснение |
| integer | SDWORD | Хранит целочисленный тип данных без знака. |
| string | BYTE | Хранит указатель на начало строки. |
| logic | SDWORD | Хранит целочисленный 0 или 1. |
| literal | BYTE  SDWORD | Литералы: символьные,  целочисленные |

## **7.3. Статическая библиотека**

В языке IMS-2020 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++.

Объявление функций статической библиотеки генерируется автоматически в коде ассемблера. Объявление функций статической библиотеки генерируется автоматически.

|  |  |
| --- | --- |
| Функция | Описание |
| void isvisokosni(int value); | Функция проверяет, является ли число, переданное параметром, високосным годом или нет, и выводит в консоль соответствующее диагностическое сообщение |
| void printcurrentdate(); | Функция выводит в консоль текущую дату и время |
| void print(char\* value); | Функция выводит в консоль строковый литерал или переменную, переданный ей через параметры |
| void printint(int value); | Функция выводит в консоль целочисленный литерал или переменную, переданный ей через параметры |
| Void concat1(char\* dest, char\* par1) | Функция выполняет присваивание строковой переменной строкового значения |
| Void concat2(char\* destination, char\* left\_operand, char\* right\_operand) | Функция выполняет конкатенацию двух строковых значений |

## **Особенности генерации кода**

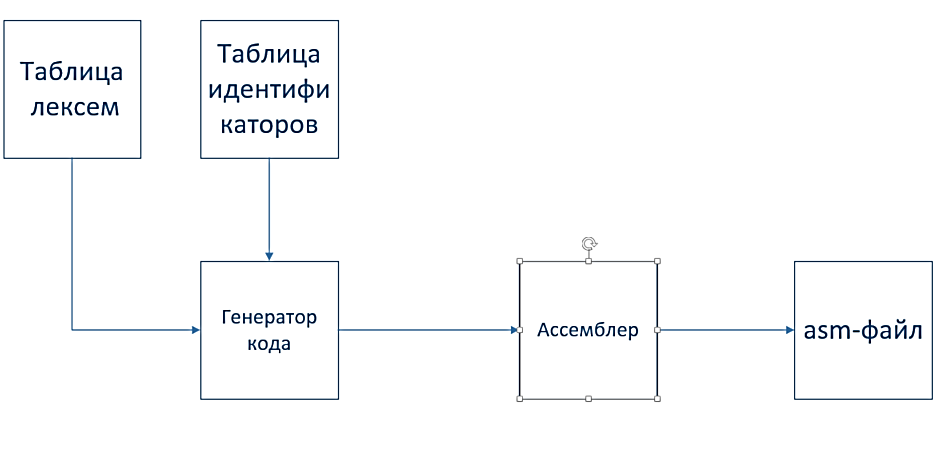
В языке IMS-2020 генерация кода строится на основе таблиц лексем и идентификаторов. Общая схема работы генератора кода представлена на рисунке 8.

Рисунок 7.2 – Структура генератора кода

## **7.5. Входные параметры генератора кода**

На вход генератору кода поступают таблицы лексем и идентификаторов исходного код программы на языке IMS-2020. 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 Е. Результат работы контрольного примера приведён на рисунке 7.3.

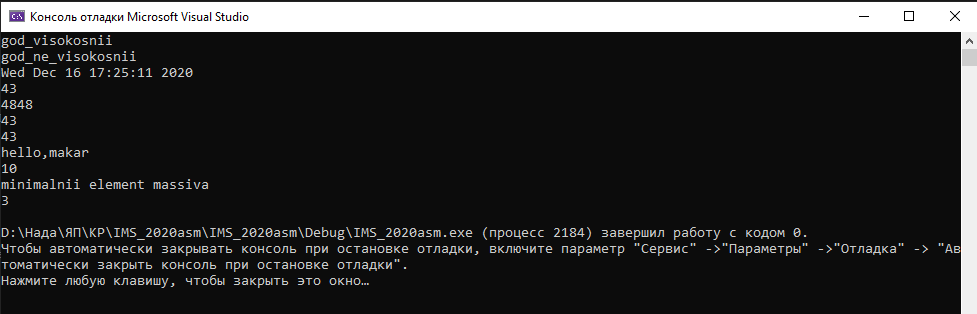


Рисунок 7.3 – Результат работы программы на языке IMS-2020

# **Глава 8. Тестирование транслятора**

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

В языке IMS-2020 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 |
| declarationё | Ошибка 111: недопустимый символ в исходном файле (-in), строка 1 символ 11 |

## **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 |
| declaration  {  integer x;  integer y;  integer y;  … | Ошибка 126: Переопределение существующего идентификатора, строка 5 символ 8 |
| declaration  {  int x;  integer y;  … | Ошибка 129: Неизвестная переменная, строка 3 символ 2 |
| declaration  {  integer x;  integer y123;  … | Ошибка 120: Нераспознанная лексема, строка 4 символ 11 |

## **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 |
| declaration  {  integer x;;  integer y;  … | 131: строка 2, Ошибочный оператор |
| declaration  {  …  integer function sum(integer a,, integer b); | 133: строка 13, Ошибка в параметрах функции |
| …  integer function sum(integer a, integer b)  {  sadc as a ++ b;  … | 132: строка 22, Ошибка в выражении |

## **8.4. Тестирование семантического анализатора**

Итоги тестирования семантического анализатора приведены в таблице 8.4.

Таблица 8.4 – Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| integer function sum(integer a, integer b)  {  …  }  …  z as x+x\*(x\*y+sum() - x);  … | Ошибка 151: количество передаваемых и принимаемых параметров не совпадают, строка 0 символ -1 |
| …  integer [6] IntegerArray 12,3,10,6,4,9;  …  print IntegerArray[12];  … | Ошибка 145: выход литерала за пределы массива, строка 0 символ -1 |
| …  integer function sum(integer a, integer b)  {  a as 12;  … | Ошибка 146: попытка присвоить значение параметру, строка 0 символ -1 |

# **Заключение**

В ходе выполнения курсовой работы был разработан транслятор и генератор кода для языка программирования IMS-2020 со всеми необходимыми компонентами. Таким образом, были выполнены основные задачи данной курсовой работы:

1. Сформулирована спецификация языка IMS-2020;
2. Разработаны конечные автоматы и важные алгоритмы на их основе для эффективной работы лексического анализатора;
3. Осуществлена программная реализация лексического анализатора, распознающего допустимые цепочки спроектированного языка;
4. Разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
5. Осуществлена программная реализация синтаксического анализатора;
6. Разработан семантический анализатор, осуществляющий проверку используемых инструкций на соответствие логическим правилам;
7. Разработан транслятор кода на язык ассемблера;
8. Проведено тестирование всех вышеперечисленных компонентов.

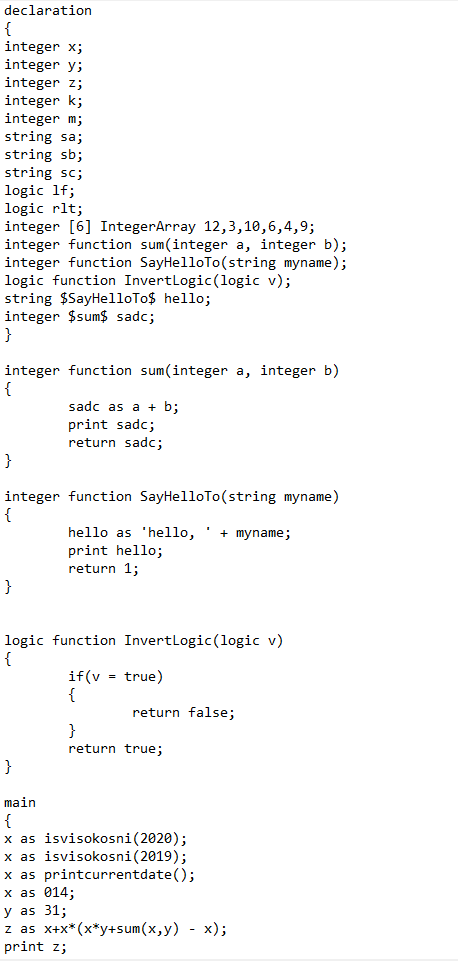
Окончательная версия языка IMS-2020 включает:

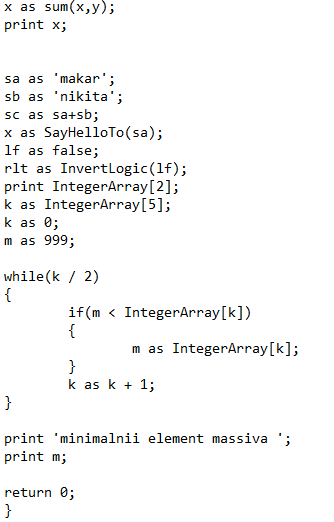
1. 3 типа данных, массивы;
2. Поддержка операторов вывода чисел и строк;
3. Возможность вызова функций стандартной библиотеки;
4. Наличие 4 арифметических и 5 логических операторов для вычисления выражений;
5. Поддержка функций, процедур, операторов цикла и условия;
6. Структурированная и классифицированная система для обработки ошибок пользователя.

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

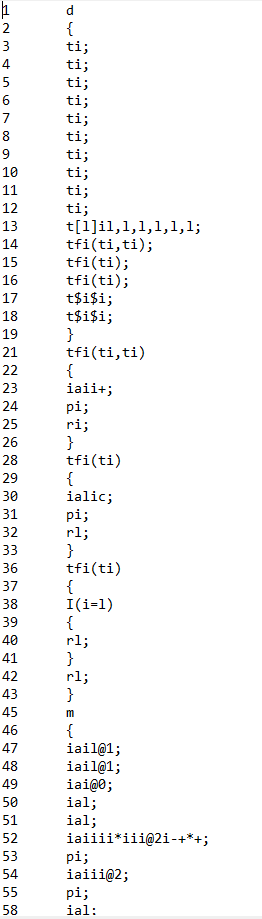
# **Приложения**

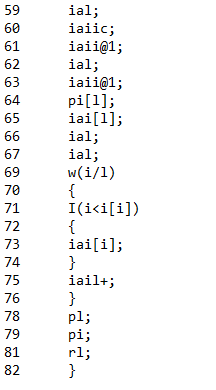
## **Контрольный пример**

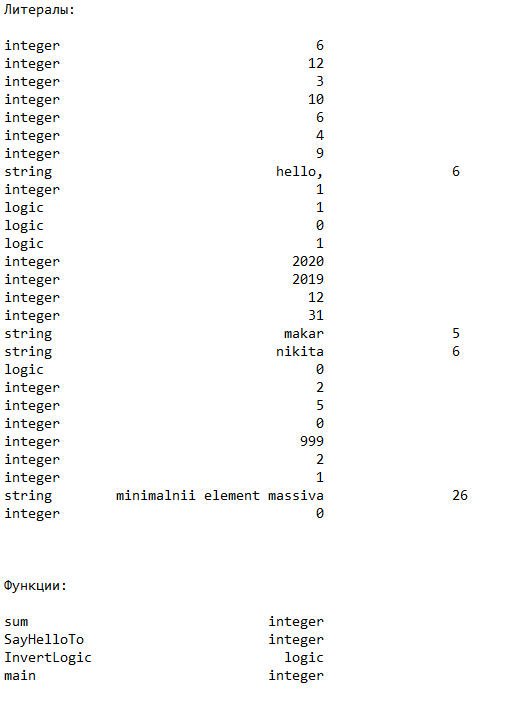


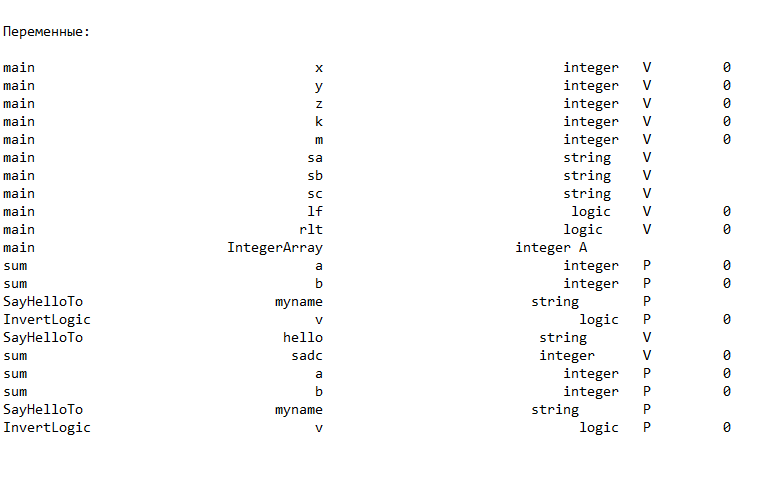


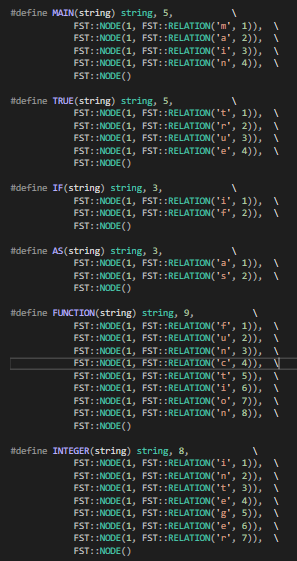
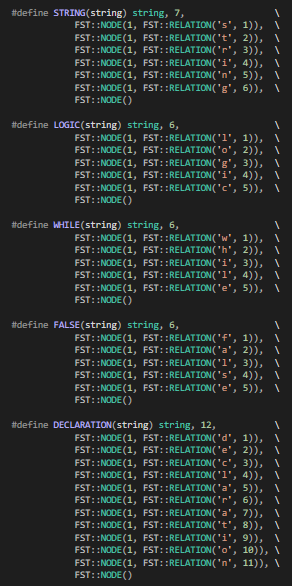
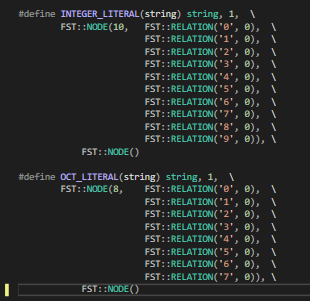
## **Приложение А**

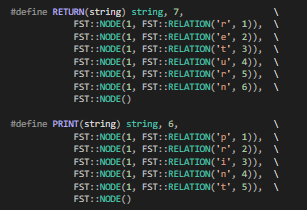
Таблица лексем:

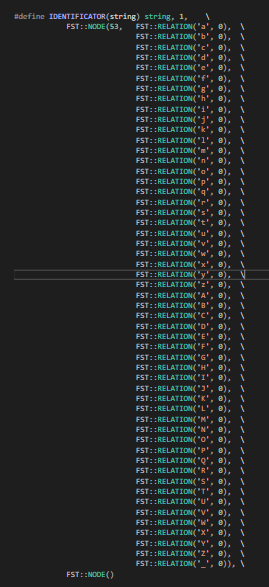
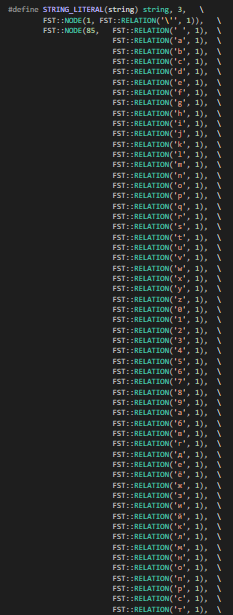


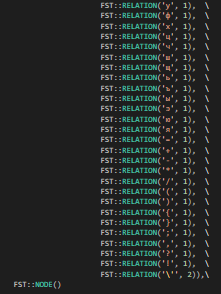
Таблица идентификаторов:



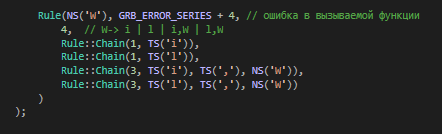
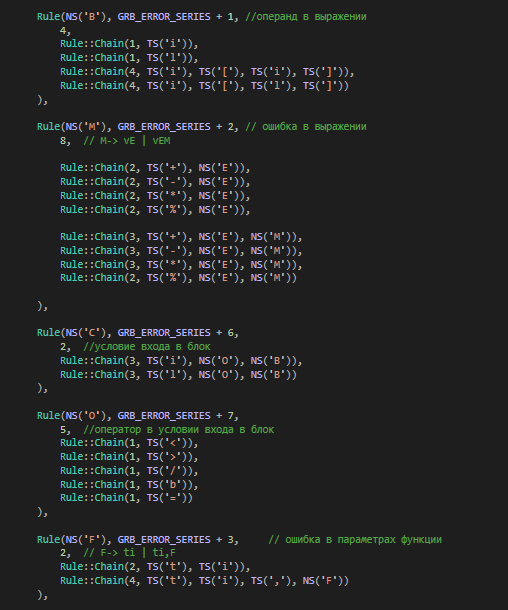








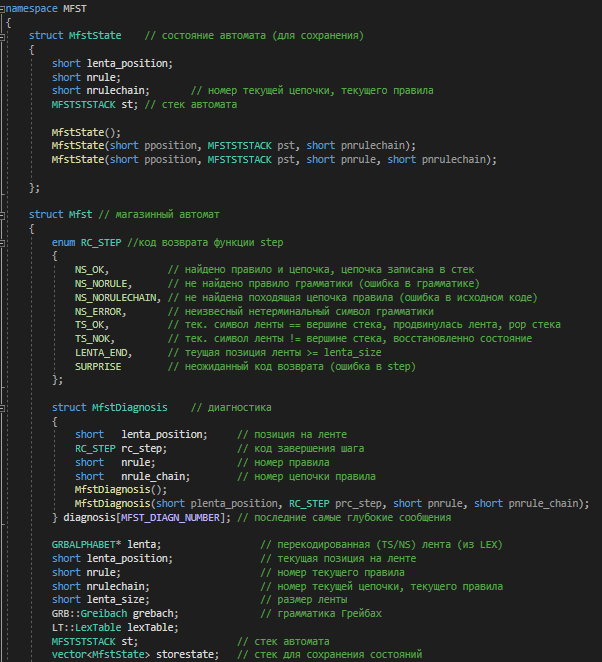
## **Приложение Б**

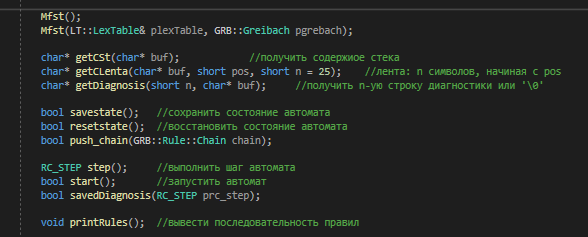


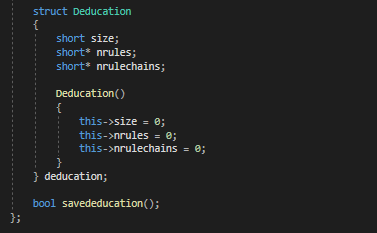
## 

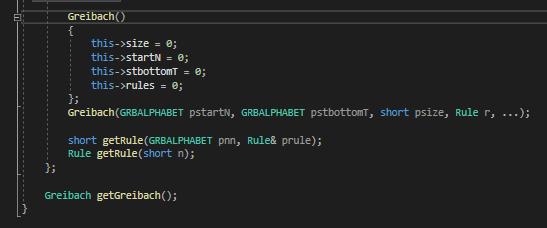
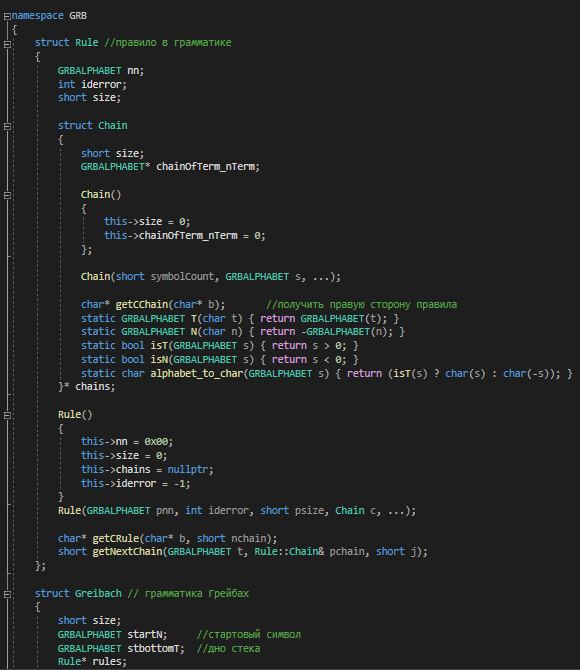
## 

## **Приложение В**



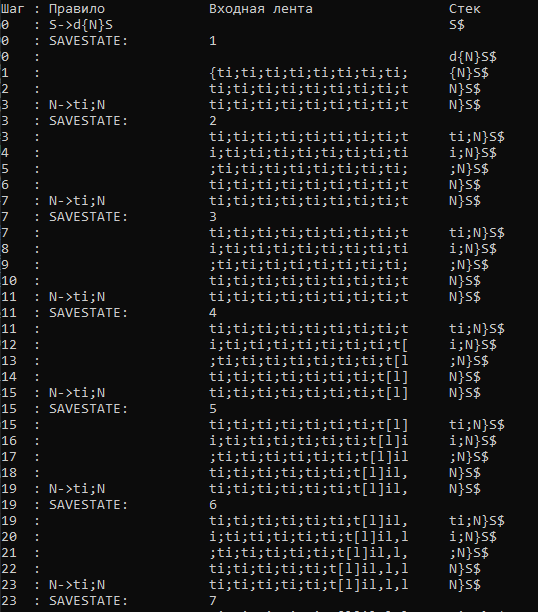




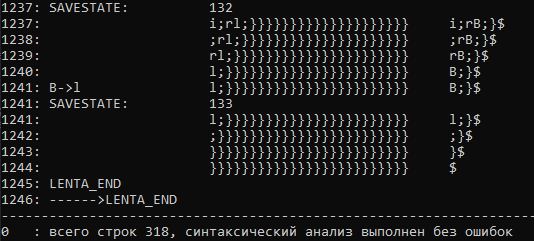


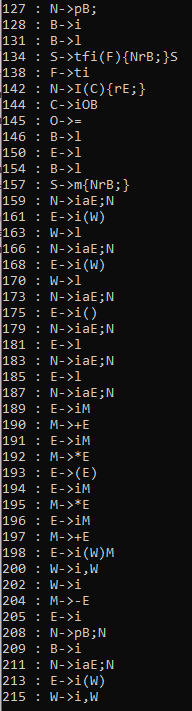
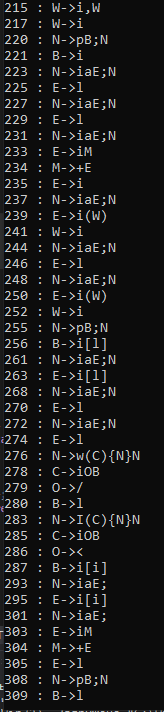
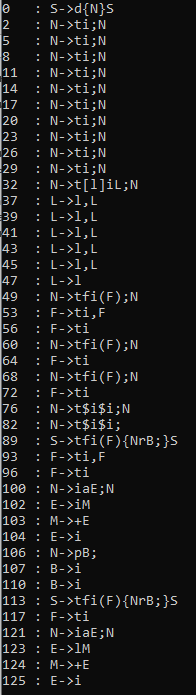
## **Приложение Г**

Начало разбора

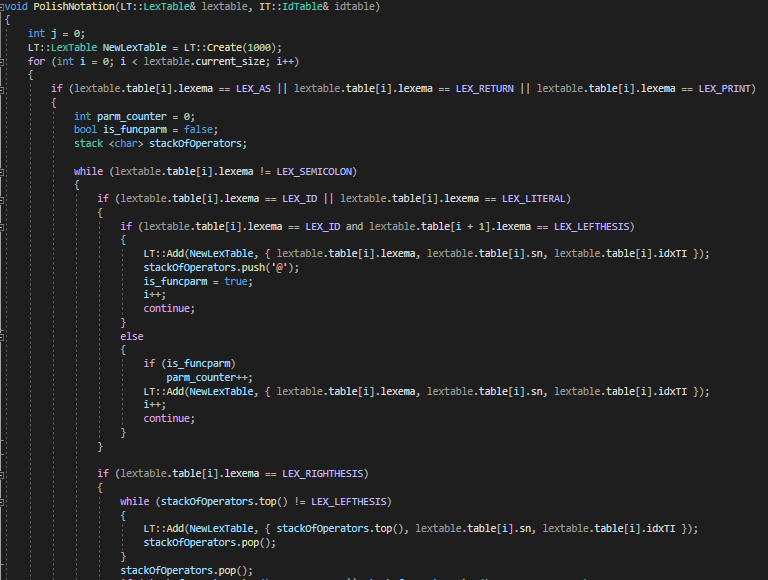


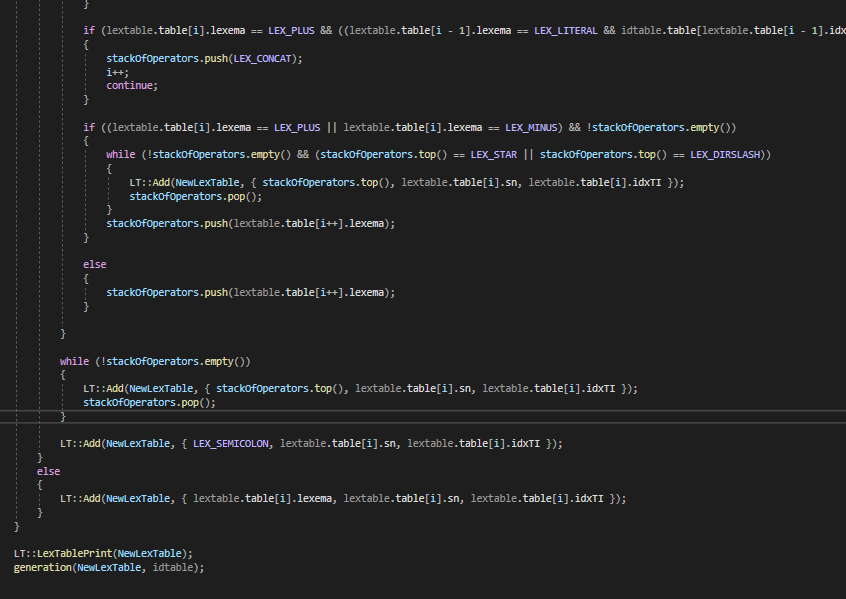
Конец разбора



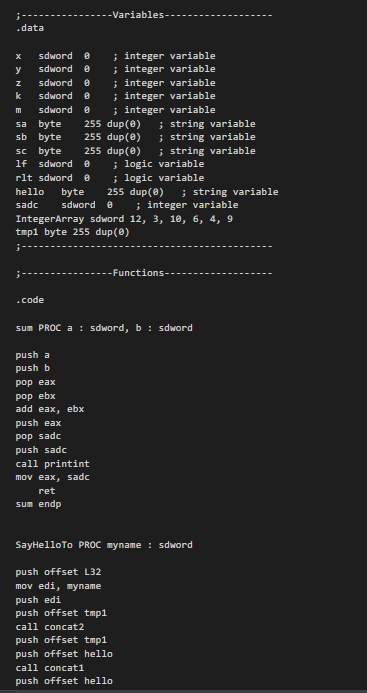
****

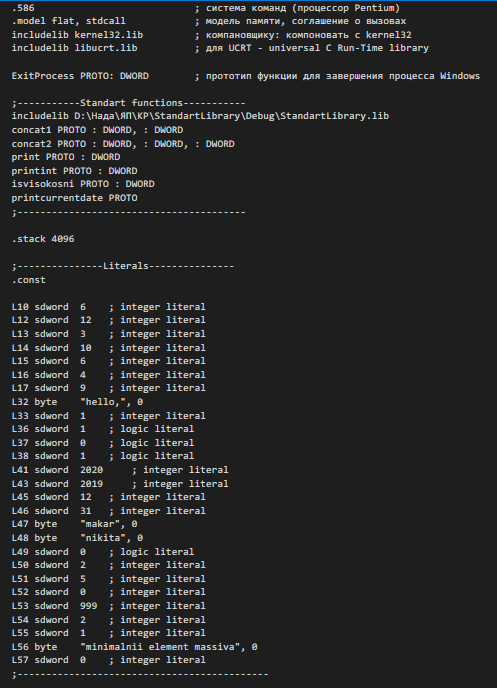
## **Приложение Д**

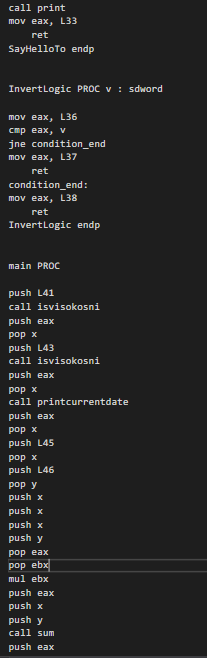
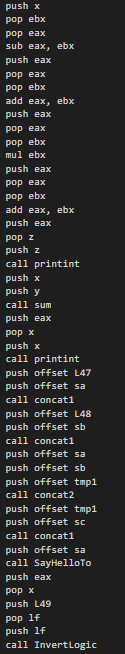
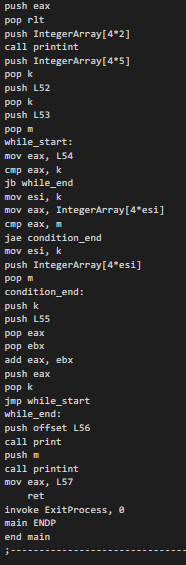




## **Приложение Е**





****

# **Литература**

1. Курс лекций по ЯП Наркевич А.С.

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

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

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

5. Страуструп, Б. Принципы и практика использования C++ / Б. Страуструп – 2009 – 1238 с