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

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора BPA-2023»

Выполнил студент Быков Павел Алексеевич

(Ф.И.О. студента)

Руководитель проекта преп.-стажер Север Александра Сергеевна

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

Заведующий кафедрой к.т.н., доц. Смелов Владимир Владиславович

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

Консультанты преп.-стажер Север Александра Сергеевна

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

Курсовой проект защищен с оценкой

Минск 2023

**Содержание**

[Введение 6](#_Toc90849301)

[1. Спецификация языка программирования 7](#_Toc90849302)

[1.1 Характеристика языка программирования 7](#_Toc90849303)

[1.2 Определение алфавита языка программирования 7](#_Toc90849304)

[1.3 Применяемые сепараторы 7](#_Toc90849305)

[1.4 Применяемые кодировки 7](#_Toc90849306)

[1.5 Типы данных 8](#_Toc90849307)

[1.6 Преобразование типов данных 8](#_Toc90849308)

[1.7 Идентификаторы 9](#_Toc90849309)

[1.8 Литералы 9](#_Toc90849310)

[1.9 Объявление данных 9](#_Toc90849311)

[1.10 Инициализация данных 9](#_Toc90849312)

[1.11 Инструкции языка 10](#_Toc90849313)

[1.12 Операции языка 10](#_Toc90849314)

[1.13 Выражения и их вычисление 11](#_Toc90849315)

[1.14 Конструкции языка 11](#_Toc90849316)

[1.15 Область видимости идентификаторов 12](#_Toc90849317)

[1.16 Семантические проверки 12](#_Toc90849318)

[1.17 Распределение оперативной памяти на этапе выполнения 12](#_Toc90849319)

[1.18 Стандартная библиотека и ее состав 12](#_Toc90849320)

[1.19 Ввод и вывод данных 13](#_Toc90849321)

[1.20 Точка входа 13](#_Toc90849322)

[1.21 Препроцессор 13](#_Toc90849323)

[1.22 Соглашение о вызовах 13](#_Toc90849324)

[1.23 Объектный код 13](#_Toc90849325)

[1.24 Классификация сообщений транслятора 13](#_Toc90849326)

[2. Структура транслятора 15](#_Toc90849327)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 15](#_Toc90849328)

[2.2 Перечень входных параметров транслятора 16](#_Toc90849329)

[2.3 Протоколы, формируемые транслятором 16](#_Toc90849330)

[3. Разработка лексического анализатора 17](#_Toc90849331)

[3.1 Структура лексического анализатора 17](#_Toc90849332)

[3.2 Контроль входных символов 17](#_Toc90849333)

[3.3 Удаление избыточных символов 18](#_Toc90849334)

[3.4 Перечень ключевых слов 18](#_Toc90849335)

[3.5 Основные структуры данных 19](#_Toc90849336)

[3.6 Структура и перечень сообщений лексического анализатора 19](#_Toc90849337)

[3.7 Принцип обработки ошибок 20](#_Toc90849338)

[3.8 Параметры лексического анализатора 20](#_Toc90849339)

[3.9 Алгоритм лексического анализа 20](#_Toc90849340)

[3.10 Контрольный пример 20](#_Toc90849341)

[4. Разработка синтаксического анализатора 21](#_Toc90849342)

[4.1 Структура синтаксического анализатора 21](#_Toc90849343)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 21](#_Toc90849344)

[4.3 Построение конечного магазинного автомата 22](#_Toc90849345)

[4.4 Основные структуры данных 23](#_Toc90849346)

[4.5 Описание алгоритма синтаксического разбора 23](#_Toc90849347)

[4.6 Структура и перечень сообщений синтаксического анализатора 24](#_Toc90849348)

[4.7 Параметры синтаксического анализатора и режимы его работы 24](#_Toc90849349)

[4.8 Принцип обработки ошибок 24](#_Toc90849350)

[4.9 Контрольный пример 24](#_Toc90849351)

[5. Разработка семантического анализатора 25](#_Toc90849352)

[5.1 Структура семантического анализатора 25](#_Toc90849353)

[5.2 Функции семантического анализатора 25](#_Toc90849354)

[5.3 Структура и перечень сообщений семантического анализатора 25](#_Toc90849355)

[5.4 Принцип обработки ошибок 26](#_Toc90849356)

[5.5 Контрольный пример 26](#_Toc90849357)

[6. Вычисление выражений 27](#_Toc90849358)

[6.1 Выражения, допускаемые языком 27](#_Toc90849359)

[6.2 Польская запись и принцип её построения 27](#_Toc90849360)

[7. Генерация кода 28](#_Toc90849361)

[7.1 Структура генератора кода 28](#_Toc90849362)

[7.2 Представление типов данных в оперативной памяти 28](#_Toc90849363)

[7.3 Статическая библиотека 29](#_Toc90849364)

[7.4 Особенности алгоритма генерации кода 29](#_Toc90849365)

[7.5 Входные параметры генератора кода 29](#_Toc90849366)

[7.6 Контрольный пример 29](#_Toc90849367)

[8. Тестирование транслятора 31](#_Toc90849368)

[8.1 Общие положения 31](#_Toc90849369)

[8.2 Результаты тестирования 31](#_Toc90849370)

[Список использованных источников 37](#_Toc90849371)

[Приложение А 38](#_Toc90849372)

[Приложение Б 40](#_Toc90849373)

[Приложение В 45](#_Toc90849374)

[Приложение Г 47](#_Toc90849375)

[Приложение Д 48](#_Toc90849376)

[Приложение Е 50](#_Toc90849377)

# **Введение**

Целью курсового проекта является разработка транслятора для языка программирования BPA-2023. Данный язык программирования предназначен для выполнения операций сравнения над числами и строками.

Исходя из цели курсового проекта, можно определить следующие задачи:

* Разработка спецификации языка программирования BPA-2023;
* Разработка лексического анализатора;
* Разработка синтаксического анализатора;
* Разработка семантического анализатора;
* Разработка генератора кода;
* Тестирование транслятора.

Пояснительная записка описывает правила и требования в использовании, принцип работы, реализацию разработанного языка программирования и компилятора.

# **1. Спецификация языка программирования**

## **1.1 Характеристика языка программирования**

Язык программирования BPA-2023 удовлетворяет требованиям процедурной парадигмы программирования. Рассматриваемый язык программирования является типизированным. Типизация языка BPA-2023 характеризуется как строгая, статическая и явная. К основным возможностям языка относиться выполнение арифметических операций, операций сравнения, работа с разными типами данных, такие как: строковые, целочисленные, логические и символьные. Кроме того язык предоставляет возможность для оперирования функциями и константами, а также возможность использования операторов ветвления и циклов.

**1.2 Определение алфавита языка программирования**

Алфавит языка определяет множество символов которые могут использоваться в файле с исходным кода языка программирования BPA-2023. Исходный код может содержать в себе символы латинского алфавита, цифры десятичной системы счисления от 0 до 9, символы пробела, табуляции, перевода строки, спецсимволы, символы операторов и символы сепараторов.

**1.3 Применяемые сепараторы**

Применяемые сепараторы в языке программирования BPA-2023, приведены в таблице 1.1.

Таблица 1.1 — Применяемые сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение сепаратора |
| ; | Разделитель инструкций |
| { } | Программный блок |
| ( ) | Параметры |
| пробел | Служит для разделения |
| , | Разделитель параметров в функции |

В контексте языка программирования BPA-2023 применяются различные сепараторы с разной логической нагрузкой.

## **1.4 Применяемые кодировки**

Для написания исходного кода на языке программирования BPA-2023 используется кодировка ASCII. Таблица кодировки ASCII представлена на рисунке 1.1.

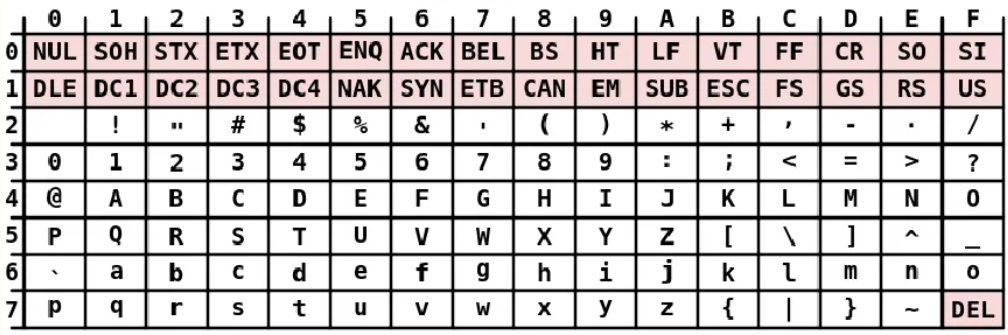


Рисунок 1.1 – Таблица кодировки ASCII

Данная кодировка используется при формировании алфавита языка.

## **1.5 Типы данных**

Описание типов данных, предусмотренных в данным языке представлено в таблице 1.2.

Таблица 1.2 – Типы данных языка BPA-2023

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| number | Фундаментальный тип данных. Предусмотрен для объявления целочисленных данных (2 байта).  Операции:  <= бинарная операция сравнения меньше либо равно;  >= бинарная операция сравнения больше либо равно;  == бинарная операция сравнения на равенство;  != бинарная операция сравнения на не равенство;  < бинарная операция сравнения меньше;  > бинарная операция сравнения больше;  + арифметическая операция сложения;  - арифметическая операция вычитания; |
| string | Является строковым типом данных. Предназначен для работы с символами, каждый символ занимает 1 байт. Максимальное количество символов – 255, включая символ окончания строки. |
| symbol | Является символьным типом данных. Предназначен для работы с символами, каждый символ занимает 1 байт. |
| bool | Является логическим типом данных. Занимает 2 байтa.  Операции:  == бинарная операция сравнения на равенство;  != бинарная операция сравнения на не равенство; |

В языке программирования BPA-2023 есть 4 типа данных: целочисленный (number), логический (bool), символьный (symbol) и строковый (string).

## **1.6 Преобразование типов данных**

В языке программирования BPA-2023 не допускается преобразование типов данных.

## **1.7 Идентификаторы**

Общее количество идентификаторов ограничено максимальным размером таблицы идентификаторов. Идентификаторы должны начинаться только с символов латинского алфавита, могут содержать цифры. Максимальная длина идентификатора равна 8 символам. Идентификаторы, объявленные внутри функционального блока, получают префикс, идентичный имени функции, внутри которой они объявлены. Префикс занимает 8 дополнительных символов. В случае превышения заданной длины, идентификаторы усекаются до длины, равной 16 символов (8 символов на имя идентификатора, 8 символов на префикс). Данные правила действуют для всех типов идентификаторов.

Идентификатор строится по данным правилам:

<буква> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<идентификатор> ::= <буква> <идентификатор> | <цифра> <идентификатор> | <буква> | <цифра>

## **1.8 Литералы**

В языке BPA-2023 существует 4 типа литералов: целый, логический, строковый и символьный типов. Их краткое описание литералов представлено в таблице 1.3.

Таблица 1.3 – Описание литералов

|  |  |
| --- | --- |
| Тип литерала | Описание |
| Литералы целого типа | Целочисленные литералы, десятичного представления. Литералы только rvalue. |
| Строковые литералы | Состоит из символов латинского алфавита, заключенных в "…" (двойные кавычки). Только rvalue. |
| Логические литералы | Состоит из одного из двух ключевых слов – true или false. Только rvalue. |
| Символьные литералы | Состоит из единственного символа латинского алфавита. Только rvalue. |

## **1.9 Объявление данных**

В языке BPA-2023 объявление переменных начинается с ключевого слова var, указывается тип данных и имя идентификатора. Требуется обязательное объявление переменной перед её использованием.

Примеры: var number a, var string b;

Объявление константы в языке BPA-2023 осуществляется при помощи ключевого слова const. По аналогии переменным, далее указывается тип данных и идентификатор. Далее, в обязательном порядке константе должно быть присвоено значение через оператора присвоения. При присвоении значения константе правостороннее выражение не может содержать вызов функции.

Примеры: const number a = 3-2, const string b = “string”;

Все переменные в языке BPA-2023 имеют область видимости, а именно префикс — название функции, в которой они находятся, что разрешает использование в различных функциях переменных с одинаковым именем. Параметры функции видны только внутри неё. Переменные, объявленные в одной функции, недоступны в другой.

## **1.10 Инициализация данных**

В момент объявления переменных происходит автоматическая инициализация в зависимости от типа данных. Инициализация другими значениями в момент объявления не допускается, за исключением объявления констант. Виды инициализации представлены в таблице 1.4.

Таблица 1.4 — Способы инициализации переменных

|  |  |
| --- | --- |
| Вид инициализации | Примечание |
| <идентификатор> = <значение>; | Присваивание переменной значения. |

## **1.11 Инструкции языка**

Инструкции языка BPA-2023 представлена в таблице 1.5.

Таблица 1.5 – Инструкции языка

|  |  |
| --- | --- |
| Инструкция | Форма записи |
| Объявление константы | const <тип данных> <идентификатор> = <значение>; |
| Объявление переменной | var <тип данных> <идентификатор> |
| Объявление внешней функции | <тип данных> function <идентификатор> ({<тип данных> <идентификатор>{,}})  {  < программный блок>  return <идентификатор>|<литерал>.  } |
| Вызов функции | <идентификатор> (<идентификатор>, …) |
| Присвоение значения | <идентификатор> = <значение>; |
| Печать данных | write <литерал>|<идентификатор> |
| Функции стандартной библиотеки | compare(string, string) — лексикографическое сравнение строк |
| Окончание таблицы 1.5 | |
| Возвращаемое значение | return <литерал>|<идентификатор> |
| Оператор цикла | repeat(<идентификатор>|<литерал>) {…} |

Представленный синтаксис строго задаёт допустимую грамматику языка.

## **1.12 Операции языка**

Операции сравнения, используются в условной конструкции, которые можно использовать в языке BPA-2023, представлены в таблице 1.6.

|  |  |
| --- | --- |
| Таблица 1.6 — Операции языка | |
| Инструкция | Форма записи |
| Операции сравнения языка | <= — бинарное меньше либо равно  == — бинарное равно  != — бинарное не равно  > — бинарное больше  < — бинарное меньше  >= — бинарное больше либо равно |
| Арифметические операции | + — бинарный плюс  - — бинарный минус |

## Таким образом, в языке BPA-2023 доступны операции сравнения и арифметический операций применимые к целочисленным данным.

## **1.13 Выражения и их вычисление**

В языке присутствуют выражения сравнения, использующиеся в условной конструкции, и арифметические операции. В выражении должны участвовать операторы и операнды совместимых типов. Все операторы имею одинаковый приоритет. Не допускается запись двух подряд арифметических операций. При записи выражений допускается вызов функций, причём как функций из стандартной библиотеки языка, так и функций определяемых пользователем.

## **1.14 Конструкции языка**

Язык BPA-2023 имеет несколько основных конструкций, правила оформления которых задают однозначную интерпретацию конечного результата, а также непосредственно влияют на семантические проверки осуществляемые в языке. Основные программные конструкции языка программирования BPA-2023 представлены в таблице 1.7.

Таблица 1.7 — Основные конструкции языка

|  |  |
| --- | --- |
| Конструкция | Реализация |
| Главная функция  (точка входа) | main  {  <программный блок>  } |
| Функции | <тип данных> function <идентификатор>  (<тип данных> <идентификатор>, …)  {  <программный блок>  return <идентификатор>|<литерал>.  } |
| Условный оператор | if(<литерал>|<идентификатор><логический оператор><литерал>|<идентификатор>)  {  <программный блок>  }  otherwise{  <программный блок>  } |
| Оператор цикла | repeat((<литерал>|<идентификатор>)  {  <программный блок>  } |

При использовании конструкций сепараторы игнорируются.

## **1.15 Область видимости идентификаторов**

Область видимости реализована по принципу «сверху вниз». В BPA-2023 требуется объявление переменной перед её использованием. Имеется возможность объявления одинаковых переменных в разных функциях. Каждая переменная получает префикс – название функции, в которой она объявлена.

## **1.16 Семантические проверки**

В языке программирования BPA-2023 выполняются следующие семантические проверки:

1. Наличие функции main – точки входа в программу;

2. Единственность точки входа;

3. Переопределение идентификаторов;

4. Использование идентификаторов без их объявления;

5. Проверка соответствия типа функции и возвращаемого параметра;

6. Правильность передаваемых в функцию параметров: количество, типы;

7. Правильность строковых выражений;

8. Превышение размера строковых и числовых литералов;

9. Правильность составленного условия цикла/условного оператора.

10. Совместимость операций для определённых типов

11. Наличия значения в переменной

12. Попытки изменения констант

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

Транслированный код использует две области памяти. В сегмент констант заносятся все литералы и константы. В сегмент данных заносятся переменные и параметры функций. Локальная область видимости в исходном коде определяется за счет использования префиксов.

## **1.18 Стандартная библиотека и ее состав**

В языке программирования BPA-2023 предусмотрена стандартная библиотека. Первоначально стандартная библиотека была написана на языке C. Далее она была собрана в статическую библиотеку с расширением lib. На этапе генерации кода библиотека автоматически подключается в файл с кодом на языке ассемблера используя директиву includelib с передачей пути к самой библиотеке в качестве параметра. Функции, входящие в состав библиотеки, описаны в табл. 1.8.

Таблица 1.8 - Функции стандартной библиотеки языка BPA-2023

|  |  |
| --- | --- |
| Функция | Описание |
| compare(string a, string b) | Данная функция целочисленного типа, принимает два строковых параметра. Сравнивает строки и возвращает 1, если равны, 0, если a меньше b, 2, если a больше b |
| pow(number a, number b) | Данная функция принимает 2 целых числа и возвращает целочисленный результат = a в степени b |

## Стандартная библиотека подключается автоматически на этапе генерации **кода. 1.19 Ввод и вывод данных**

Ввод данных языком программирования BPA-2023 не поддерживается.

Для вывода данных используется функция write(<имя идентификатора>|<литерал>).

Примеры: write(a), write(2);

Команда write реализуется через заранее написанные библиотечные функции. В зависимости от типа принимаемого параметра будет вызвана определенная функция.

## **1.20 Точка входа**

Функция точки входа в языке программирования BPA-2023 представлена в таблице 1.9.

Таблица 1.9 — Точка входа

|  |  |
| --- | --- |
| Конструкция | Реализация |
| Главная функция  (точка входа) | main  {  <программный блок>  } |

Точка входа обозначается ключевым словом main.

## **1.21 Препроцессор**

В языке программирования BPA-2023 препроцессор отсутствует.

## **1.22 Соглашение о вызовах**

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

– все параметры функции передаются через стек;

– память высвобождает вызываемый код;

– занесение в стек параметров идёт справа налево

## **1.23 Объектный код**

Язык программирования BPA-2023 транслируется в язык ассемблера MASM.

## **1.24 Классификация сообщений транслятора**

Генерируемые транслятором сообщения определяют степень его информативности, то есть сообщения транслятора должны давать максимально полную информацию о допущенной пользователем ошибке при написании программы. Каждая ошибка также содержит информацию о том, на каком этапе она была вызвана; именно этот параметр характеризует разбиение диапазонов кодов ошибок.

Сообщения транслятора приведены в таблице 1.10, а также в приложении Б.

Таблица 1.10 Классификация ошибок(диапазон)

|  |  |
| --- | --- |
| Префикс ошибки | Описание ошибки |
| Ошибка ###: [Системная]: | ### - код ошибки. Сообщение, генерируемое при критической ошибке. Диапазон: 0-99 |
| Ошибка ###: [Лексическая]: | ### - код ошибки. Сообщение, генерируемое на этапе лексического анализа. Диапазон: 200-299 |
| Ошибка ###: [Синтаксическая]: | ### - код ошибки. Сообщение, генерируемое на этапе синтаксического анализа. Диапазон: 300-399 |
| Ошибка ###: [Семантическая]: | ### - код ошибки. Сообщение, генерируемое на этапе семантического анализа. Диапазон: 400-499 |

**1.25 Контрольный пример**

Контрольный пример представлен в приложении А.

# **2. Структура транслятора**

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

Задачей транслятора является преобразование программы, написанной на языке BPA-2023, 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

Первая стадия работы компилятора называется лексическим анализом, а программа, её реализующая, – лексическим анализатором (сканером). Назначением лексического анализатора является нахождение ошибок лексики языка и формирование таблицы лексем и таблицы идентификаторов. Подробнее описан в 3 главе.

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

Семантический 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.

Генератор кода – часть 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 BPA-2023, 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, представленные в таблице 2.1.

Таблица 2.1 Входные параметры транслятора языка BPA-2023

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание параметра | Значение по умолчанию |
| -in:<путь к in-файлу> | Файл с исходным кодом на языке BPA-2023, имеющий расширение .txt | Не предусмотрено |
| -log:<путь к log-файлу> | Файл журнала для вывода протоколов работы программы. | Значение по умолчанию:  <имя in-файла>.log |
| -out:<путь к out-файлу> | Выходной файл – результат работы транслятора. Содержит исходный код на языке асемблера. | Значение по умолчанию:  <имя in-файла>.asm |

## Допускается передача параметров в программу посредством консоли.

## **2.3 Протоколы, формируемые транслятором**

Таблица с перечнем протоколов, формируемых транслятором BPA-2023 и их назначением представлена в таблице 2.2

Таблица 2.2 Протоколы, формируемые транслятором языкаBPA-2023

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |
| Файл журнала, заданный параметром "-log:" | Файл с протоколом работы транслятора языка программирования BPA-2023. Содержит таблицу лексем и таблицу идентификаторов |
| Выходной файл, заданный параметром "-out:" | Результат работы программы – файл, содержащий исходный код на языке ассемблера. |

Расположение все протокол опционально может быть настроено ещё при вызове программы. Для этого используются параметры приведенные в таблице.

# **3. Разработка лексического анализатора**

## **3.1 Структура лексического анализатора**

Лексический анализатор – часть транслятора, выполняющая лексический анализ. Лексический анализатор преобразует исходный текст, заменяя лексические единицы их внутренним представлением – лексемами, для создания промежуточного представления исходной программы. Каждой лексеме сопоставляется ее тип и запись в таблице идентификаторов, в которой хранится дополнительная информация.

Функции лексического анализатора:

− удаление «пустых» символов.

− распознавание идентификаторов и ключевых слов;

− распознавание констант;

− распознавание разделителей и знаков операций.

На выходе формируется таблица лексем и таблица идентификаторов. Структура лексического анализатора представлена на рисунке 3.1



Рисунок 3.1 Структура лексического анализатора BPA-2023

В последующем таблицы используются на других этапа анализа языка.

## **3.2 Контроль входных символов**

Таблица для контроля входных символов представлена на рисунке 3.2.

Принцип работы таблицы заключается в получении кодового значения каждого символа входной последовательности согласно установленной кодировке.

Далее код данного символа сопоставляется с таблицей, таким образом мы получаем информацию о том, к какой категории относиться данный символ.

Далее приведён пример возможных значений. Описание значения символов: T – разрешённый символ, F – запрещённый символ, S – сепаратор.

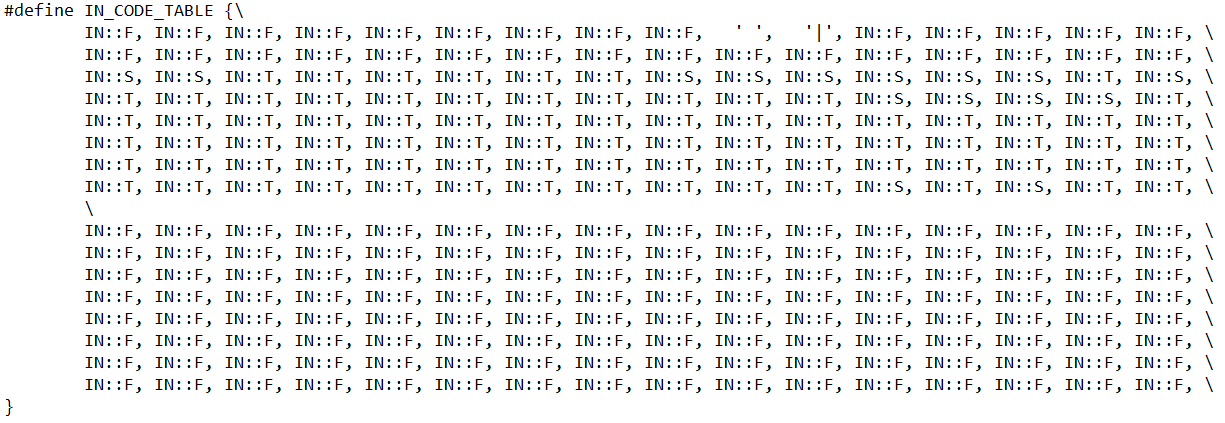


Рисунок 3.2. Таблица контроля входных символов

## **3.3 Удаление избыточных символов**

Избыточными символами являются символы табуляции и пробелы. Табуляция заменяется на пробел. Пробелы являются сепараторами.

Описание алгоритма удаления избыточных символов:

1. Посимвольно считываем файл с исходным кодом программы.

2. Пробелы выступаю в роли сепаратора и пропускаются.

## **3.4 Перечень ключевых слов**

Лексемы – это символы, соответствующие ключевым словам, символам операций и сепараторам, необходимые для упрощения дальнейшей обработки исходного кода программы. Таким образом, дальнейшая работа с данными будет сводиться к работе с таблицами лексем и таблицой идентификаторов.

Данное соответствие описано в таблице 3.1.

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

|  |  |  |
| --- | --- | --- |
| Тип | Слова | Лексема |
| Ключевые слова | include | u |
| number | t |
| string | t |
| function | f |
| write | p |
| pow | b |
| compare | a |
| return | r |
| main | m |
| Продолжение таблицы 3.1 | | |
| Тип | Слова | Лексема |
| Ключевые слова | if | c |
| otherwise | e |
| repeat | y |
| var | d |
| const | k |
| symbol | t |
| bool | t |
| Иное | Идентификатор | i |
| Целочисленный литерал | l |
| Строковый литерал | L |
| Сепараторы | ; | ; |
| , | , |
| { | { |
| } | } |
| ( | ( |
| ) | ) |
| Операторы | = | = |
| + | + |
| - | - |
| Логические операторы | o |

Пример реализации таблицы лексем представлен в приложении Б.

Также в приложении Б находятся некоторые конечные автоматы, соответствующие лексемам языка BPA-2023.

## **3.5 Основные структуры данных**

К основным структура данных которыми оперирует лексический анализатор относится таблица лексем, а также таблица идентификаторов. Кроме того, лексический анализатор работает с конечными автоматами. Структуры таблиц лексем и идентификаторов данных языка BPA-2023, используемых для хранения, представлены в приложении В. В таблице лексем содержится лексема, её номер, полученный при разборе, номер строки в исходном коде. В таблице идентификаторов содержится имя идентификатора, номер в таблице лексем, тип данных, смысловой тип идентификатора, флаг определенности значения и его значение. Кроме того, во внутренней структуре программы, а также внутренних алгоритмах транслятора иногда использовались структуры из стандартной библиотеки языка программирования C++ такие как queue, stack, map.

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

Перечень сообщений лексического анализатора представлен в таблице 3.2.

Таблица 3.2 – Перечень ошибок лексического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение об ошибки |
| 200 | Двойное объявление |
| 201 | Ошибка лексического анализа |
| 202 | Длина идентификатора |
| 203 | Идентификатор не определён |
| 204 | Максимальный размер таблицы идентификаторов 4096 |
| 205 | Таблица идентификаторов переполнена |
| 206 | Максимальный размер таблицы лексем 4096 |
| 207 | Таблица лексем переполнена |
| 208 | Максимальное значение number 32767 |
| 209 | Превышен максимальный размер symbol |

Кроме того, все вышеперечисленные ошибки имею префикс – “[Лексическая]”.

## **3.7 Принцип обработки ошибок**

При возникновении ошибки транслятор завершает свою работу. Для обработки ошибок лексический анализатор использует таблицу с сообщениями. Структура сообщений содержит информацию о номере сообщения, номер строки и позицию, где было вызвано сообщение в исходном коде, информацию об ошибке.

## **3.8 Параметры лексического анализатора**

Входным параметром лексического анализа является структура, полученная после чтения входного файла на этапе проверки исходного кода на допустимость символов.

## **3.9 Алгоритм лексического анализа**

Алгоритм работы лексического анализа:

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

Распознавание цепочек основывается на работе конечных автоматов. Работу конечного автомата можно проиллюстрировать с помощью графа переходов.

Пример. Регулярное выражение для ключевого слова «main».

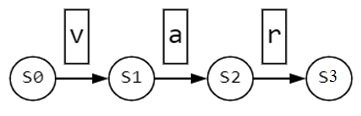


Рисунок 3.3 – Граф переходов для цепочки ‘main’

Граф конечного автомата для этой лексемы представлен на рисунке 3.3. S0 – начальное состояние, S3 – конечное состояние автомата.

## **3.10 Контрольный пример**

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

# **4. Разработка синтаксического анализатора**

## **4.1 Структура синтаксического анализатора**

Синтаксический анализ – это фаза компилятора, 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 BPA-2023 используется контекстно-свободная грамматика , где

T – множество терминальных символов (алфавит языка BPA-2023),

N – множество нетерминальных символов,

P – множество правил языка,

S – начальный символ грамматики, являющийся нетерминалом.

Эта грамматика приведена к нормальной форме Грейбах, 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 BPA-2023 представлена в приложении Г.

TS – терминальные символы, которыми являются сепараторы, знаки арифметических операций и некоторые строчные буквы, которые приведены в разделе 1.2

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

Таблица 4.1 – Перечень правил, составляющих грамматику языка и описание нетерминальных символовBPA-2023

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| S | S→m{NrE;};| tfi(F){NrE;};S| tfi( ){NrE;};S| ul | ulS | Порождает правила, описывающее общую структуру программы |
| N | N→dti;|i=E;|dti;N| i=E;N|p(l);N| p(i);N|p(l);| p(i);|  c(C){N}e{N}N| c(C){N}e{N}|  c(C){N}| c(C){N}N | Порождает правила, описывающие инструкции языка |
| E | E→i|l|i(W)|i( )| b(W)|a(W)| i+l|i+i|l+l|i-i|i-l|l-l | Порождает правила, описывающие выражения |
| F | F→ti|ti,F | Порождает правила, описывающие параметры локальной функции при её объявлении |
| W | W→i|l|i,W|l,W | Порождает правила, описывающие параметры вызываемой функции |
| C | С→ioi|iol|loi|lol | Порождает правила, описывающие условия оператора if |

## **4.3 Построение конечного магазинного автомата**

Конечный автомат с магазинной памятью представляет собой семерку, описание которой представлено в таблице 4.2.

Таблица 4.2 – Описание компонентов магазинного автомата

|  |  |  |
| --- | --- | --- |
| Компонента | Определение | Описание |
|  | Множество состояний автомата | Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата |
|  | Алфавит входных символов | Алфавит является множеством терминальных и нетерминальных символов, описание которых содержится в разделе 1.2 и в таблице 4.1. |

Продолжение таблицы 4.2

|  |  |  |
| --- | --- | --- |
| Компонента | Определение | Описание |
|  | 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 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 представлены в виде структуры магазинного конечного автомата, выполняющего разбор исходной ленты, и структуры грамматики Грейбах, описывающей синтаксические правила и цепочки правил. Данные структуры представлены в приложении Г.

## **4.5 Описание алгоритма синтаксического разбора**

Принцип работы автомата:

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

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

Перечень сообщений синтаксического анализатора представлен на рисунке 4.1.

Таблица 4.3 – Перечень ошибок синтаксического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение ошибки |
| 300 | Неверная структура программы |
| 301 | Ошибочная конструкция в функции |
| 302 | Ошибка в выражении |
| 303 | Ошибка в параметрах функции |
| 304 | Ошибка в параметрах вызываемой функции |
| 305 | Ошибка в условии |
| 306 | Ошибка в операторе |

Алгоритмы синтаксического анализа основа на грамматике Грейбах и выдаёт ошибки в зависимости от выбранного набора правил. Кроме того, все вышеперечисленные ошибки имею префикс – “[Синтаксическая]”.

## **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 правила разбора, которые выводятся на консоль.

## **4.8 Принцип обработки ошибок**

Синтаксический анализатор выполняет разбор исходной последовательности лексем до тех пор, пока не дойдёт до конца цепочки лексем или не найдёт ошибку. Тогда анализ останавливается и выводится сообщение об ошибке, если она найдена. Если в процессе анализа находятся более трёх ошибок, то анализ останавливается. После всей процедуры трассировки в протокол будет выведено диагностическое сообщение.

## **4.9 Контрольный пример**

Пример разбора синтаксическим анализатором исходного кода на языке BPA-2023 представлен в приложении Д. Дерево разбора исходного кода также представлено в графическом материале.

# **5. Разработка семантического анализатора**

## **5.1 Структура семантического анализатора**

Семантический анализ языка BPA-2023 выполняется после выполнения лексического и синтаксического анализа. Несмотря на это, некоторые семантические проверки выполняются на этапе лексического анализа. На вход семантического анализатора подаются таблица лексем и таблица идентификаторов. Схема семантического анализатора изображена на рисунке 5.1.

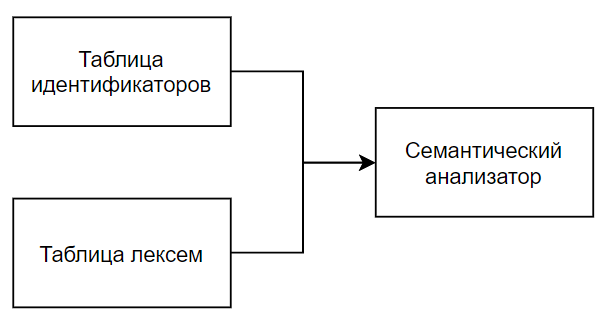


Рисунок 5.1 Схема семантического анализатора языка BPA-2023

## **5.2 Функции семантического анализатора**

Семантический анализатор выполняет проверку на основные правила языка (семантики языка), которые описаны в разделе 1.16. Прежде всего эти проверки связаны со сверкой допустимых типов данных и типов идентификаторов. Таким образом, семантический анализ при обработке каждой лексемы, записанной в таблицу лексем рассматривается как отдельный случай и применяются отдельные правила. Эти правила обусловлены прежде всего смысловой нагрузкой языка программирования BPA-2023. Кроме того, некоторые проверки предусматривают дополнительные проверки на внутреннем уровне, например при обработке идентификаторов отдельно выполняется ветка кода ответственная за обработку вызовов функций, а та, в свою очередь, отдельную ветку за обработку библиотечных функций.

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

Сообщения, формируемые семантическим анализатором, представлены в таблице 5.1.

Таблица 5.1 – сообщения ошибок семантического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение об ошибки |
| 400 | Нет точки входа main |
| 401 | Более одной точки входа main |
| 402 | Возвращаемое значение не соответсвует возвращаемому типу функции |
| 403 | Количество параметров функции больше допустимого |
| 404 | Передаваемые параметры не соответствуют функции |
| 405 | Не соответсвует количество передаваемых параметров |
| 406 | Ошибка в параметрах встроенной функции |
| 407 | Ошибка в параметрах функции compare() |
| 408 | Несовместимый оператор и тип данных |
| 409 | Ошибка в параметре функции write() |
| 410 | Ошибка в подключении библиотеки |
| 411 | Ошибка в параметрах цикла repeat() |
| 412 | Недопустимое сочитание литерала и типа данных |
| 413 | Попытка изменить значение константы |
| 414 | Попытка использования переменной без значения |
| 415 | Значение константы не может зависеть от результата функции |

Структура сообщений семантического анализатора идентична сообщениям ошибок этапа лексического анализа (раздел 3.6). Кроме того, все вышеперечисленные ошибки имею префикс – “[Семантическая]”.

## **5.4 Принцип обработки ошибок**

Главной особенностью обработки ошибок семантического анализатора заключается в акценте на контекст применяемых конструкций. В основном проверки начинаются вокруг операторов, получаю значения лексем применяемых к этому оператору. Таким образом проверяются совместимость значений идентификатора привязанного к этим лексемам.

## **5.5 Контрольный пример**

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

Таблица 5.1. Примеры диагностики ошибок

|  |  |
| --- | --- |
| Текст ошибки | Исходный код |
| Ошибка 401: [Cемантическая]: Более одной точки входа main, строка 0, позиция 0 | main  {  …  }  main  {  …  } |
| Ошибка 402: [Cемантическая]: Возвращаемое значение не соответствует типу функции, строка 2, позиция 0 | string function fun(number a){  return a;  } |
| Ошибка 404: [Cемантическая]: Передаваемые параметры не соответствуют функции, строка 7, позиция 0 | string function fun(number a){  return a;  }  main  {  var string str;  str = func("b");  return 0;  } |

# **6. Вычисление выражений**

## **6.1 Выражения, допускаемые языком**

В языке BPA-2023 допускаются выражения, применимые к целочисленным типам данных. В выражениях поддерживаются логические операции, такие как < > == <= >= != и арифметические +-

## **6.2 Польская запись и принцип её построения**

В языке BPA-2023 польская запись не используется.

# **7. Генерация кода**

## **7.1 Структура генератора кода**

На вход генератора подаются таблицы лексем и идентификаторов, на основе которых генерируется файл с ассемблерным кодом.

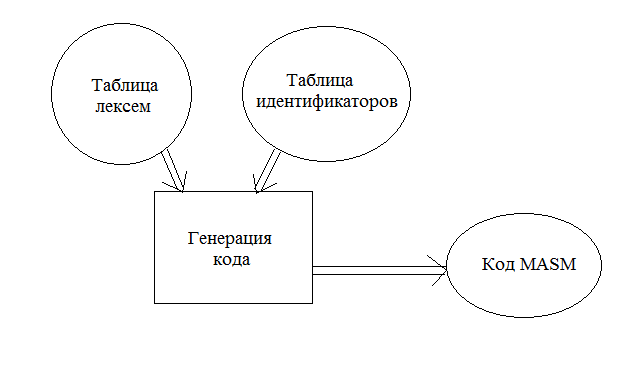


Рисунок 7.1 Структура генератора кода

Генерация кода языка BPA-2023 осуществляется в язык ассемблера.

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

Соответствия между типами данных на языке BPA-2023 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов языка BPA-2023 и языка Ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языкеBPA-2023 | Тип идентификатора на языке ассемблера | Пояснение |
| number | WORD | Хранит целочисленный тип данных со знаком. |
| string | DWORD | Хранит указатель на начало строки. |
| bool | WORD | Хранит целочисленный тип |
| symbol | BYTE | Хранит единственный символ |
| L(0-99) | BYTE  WORD | Литералы: символьные,  целочисленные |

Идентификаторы языка BPA-2023 размещены в сегменте данных(.data). Литералы – в сегменте констант (.const).

## **7.3 Статическая библиотека**

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

Таблица 7.3 – Функции статической библиотеки

|  |  |
| --- | --- |
| Функция | Назначение |
| \_pow(number a, number b) | Возводит число a в степень b |
| \_writeN(number a)  \_writeS(string str) | Вывод на консоль целочисленной переменной a и строки str |
| \_compare(string a, string b) | Лексикографическое сравнение строк, если строка a меньше b возвращает 0, равна 1, больше 2 |

## **7.4 Особенности алгоритма генерации кода**

В языке BPA-2023 генерация кода строится на основе таблиц лексем и идентификаторов. Преобразования происходят по мере прохождения по таблицам. Функции статической библиотеки начинаются с нижнего подчёркивания для исключения их переопределения. Перед началом основной трансляции производится запись литералов и идентификаторов в сегменты констант и данных соответственно.

## **7.5 Входные параметры генератора кода**

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

## **7.6 Контрольный пример**

Результат выполнения сгенерированного ассемблерного кода на основе контрольного примера приведен в приложении Е. Результат работы контрольного примера приведён на рисунке 7.1.

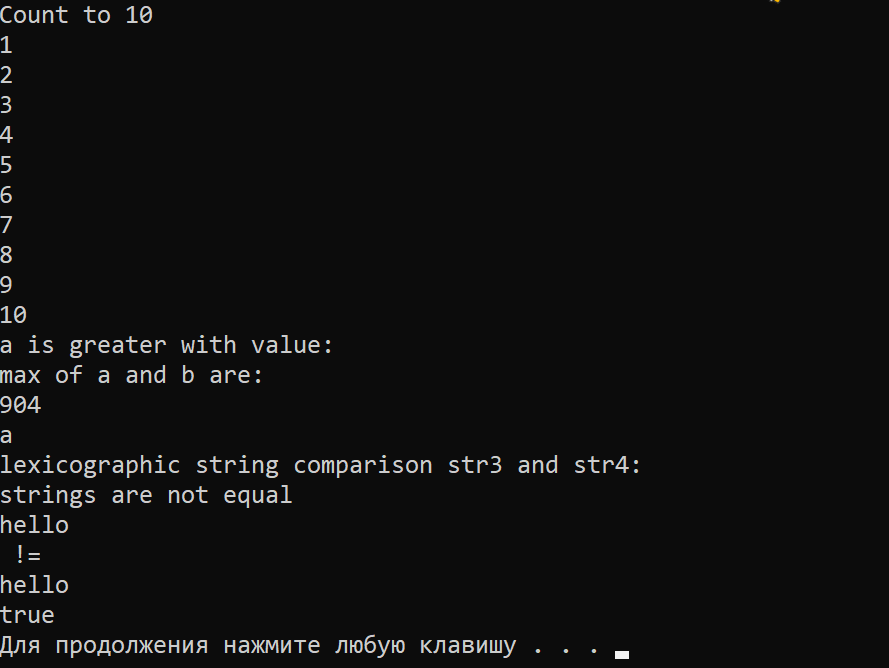


Рисунок 7.1 Результат работы программы на языке BPA-2023

После выполнения также срабатывает команда system(“pause”), таким образом программа не закроется раньше времени.

# **8. Тестирование транслятора**

## **8.1 Общие положения**

При возникновении ошибки на каком-либо этапе трансляции, она обрабатывается в главном файле программы: ошибка выводится на консоль и записывается в файл логирования.

## **8.2 Результаты тестирования**

Описание тестовых наборов, демонстрирующих проверки на разных этапах трансляции, приведено в таблице 8.1

Таблица 8.1 – Результаты тестирования транслятора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| буква а | Ошибка 111: [Системная]: Недопустимый символ в исходном файле (-in), строка 1, позиция 0 |
| main  {  var string str1;  var string str1;  } | Ошибка 200: [Лексическая]: Двойное объявление, строка 4, позиция 0 |
| main  {  A  } | Ошибка 201: [Лексическая]: Ошибка лексического анализа, строка 3, позиция 0 |
| var string a23456789; | Ошибка 202: [Лексическая]: Длина идентификатора больше 8, строка 1, позиция 0 |
| str = ""; | Ошибка 203: [Лексическая]: Идентификатор не определён, строка 1, позиция 0 |
| main  {  var number a;  a = 100000;  } | Ошибка 208: [Лексическая]: Максимальное значение типа number 32767, строка 4, позиция 0 |
| var number a; | Ошибка 400: [Cемантическая]: Нет точки входа main, строка 0, позиция 0 |

Продолжение таблицы 8.1

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  {  }  main  {  } | Ошибка 401: [Cемантическая]: Более одной точки входа main, строка 0, позиция 0 |
| string function func()  {  var number a;  return 0;  }  main  {  var number a;  return 0;  } | Ошибка 402: [Cемантическая]: Возвращаемое значение не соответствует типу функции, строка 4, позиция 0 |
| number function func(number a, number b, number c, number d, number e, number f)  {  a = 3;  return a;  }  main  {  var number a;  a = func(1, 2, 3, 4, 5, 6);  return 0;  } | Ошибка 403: [Cемантическая]: Количество параметров функции больше допустимого, строка 9, позиция 0 |

Продолжение таблицы 8.1

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| number function func(number a)  {  a = 3;  return a;  }  main  {  var number a;  a = func("1");  return 0;  } | Ошибка 404: [Cемантическая]: Передаваемые параметры не соответствуют функции, строка 9, позиция 0 |
| number function func(number a)  {  a = 3;  return a;  }  main  {  var number a;  a = func(1, 2);  return 0;  } | Ошибка 405: [Cемантическая]: Не соответсвует количество передаваемых параметров, строка 9, позиция 0 |
| uselib "../Debug/StaticLib.lib"  main  {  var number a;  a = pow(1);  return 0;  } | Ошибка 406: [Cемантическая]: Ошибка в параметрах функции pow(), строка 5, позиция 0 |

Продолжение таблицы 8.1

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| include …/BPA-lib.lib"  main  {  var number a;  a = compare(1);  return 0;  } | Ошибка 407: [Cемантическая]: Ошибка в параметрах функции compare(), строка 5, позиция 0 |
| main  {  var number a;  var string b;  if(a > b){  write("some");  }  return 0;  } | Ошибка 408: [Cемантическая]: Попытка сравнения операндов не типа number, строка 5, позиция 0 |
| include 5  main  {  write("a");  return 0;  } | Ошибка 410: [Cемантическая]: Ошибка в подключении библиотеки, строка 1, позиция 0 |
| number function func(number a)  {  var string;  return 0;  }  main  {  var string a;  return 0;  } | 301: строка 1, [Cинтаксическая] : Ошибочная конструкция в функции |

Продолжение таблицы 8.1

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  {  var string a;  a = 1, 2;  return 0;  } | 302: строка 4, [Cинтаксическая]: Ошибка в выражении |
| number function func(number a)  {  a = 0;  return a;  }  main  {  var number a;  a = func(,);  return 0;  } | 304: строка 9, [Cинтаксическая]: Ошибка в параметрах вызываемой функции |
| number function func(,)  {  a = 0;  return a;  }  main  {  var number a;  a = func(1);  return 0;  } | 303: строка 1, [Cинтаксическая]: Ошибка в параметрах функции |
| main  {  if("a")  {  write("a");  }  return 0;  } | 305: строка 3, [Cинтаксическая]: Ошибка в условии |

**Заключение**

В ходе выполнения курсовой работы был разработан транслятор для языка программирования BPA-2023. Таким образом, были выполнены основные задачи данной курсовой работы:

* Представлена спецификация языка BPA-2023;
* Разработан лексический анализатор;
* Разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика;
* Разработан синтаксический анализатор;
* Разработан семантический анализатор;
* Разработан транслятор кода на язык ассемблера;
* Проведено тестирование транслятора.

Окончательная версия языка BPA-2023 включает:

* 4 типа данных;
* Возможность оперирования константами
* Возможность вызова функций стандартной библиотеки;
* Наличие 6 логических операторов и 2 арифметических операций;
* Поддержку функций, оператора условия и цикла;
* Возможность оставлять комментарии в коде;

# **Список использованных источников**

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

2. Вирт, Н. Построение компиляторов/ Никлаус Вирт. - ДМК-Пресс – Москва, 2016. - 188 с.

3. Ирвин, К. Р. Язык ассемблера для процессоров Intel / К. Р. Ирвин. – M.: Вильямс, 2005. – 912с.

# **Приложение А**

|  |
| --- |
| Контрольный пример |
| include "D:/LABS2/BPA-2023/BPA-lib/Debug/BPA-lib.lib"  number function max(number a, number b){  var number res;  if (b > a) {  res = b;  } otherwise {  res = a;  }  return res;  }  main  {  var number d;  var number a;  a = 10;  const number cos = 5 + a - 3;    a = 1000 - 100 + pow(2, 2);  var number b;  b = 1;  write("Count to 10");  repeat(10)  {  write(b);  b = b + 1;  }  if(b >= a){  write("b is greater with value: ");  } otherwise {  write("a is greater with value: ");  }  if(a != b){  write("max of a and b are: ");  d = max(a, b);  write(d);  }  const symbol lettera = 'a';  write(lettera);  var string str3;  var string str4;  write("lexicographic string comparison str3 and str4:");  str3 = "hello";  str4 = "hello ";  var number res;  res = compare(str3, str4);  write(str3);  if(res == 1) {  write(" == ");  } otherwise {  write(" != ");  }  write(str4);  var bool info;  info = true;  if(info == true){  write("true");  }  return 0; } |

Листинг А.1 – Исходный код контрольного примера

# **Приложение Б**

Результат работы лексического анализа

|  |
| --- |
| 01 ul<0>  04 tfi<1>(ti<2>,ti<3>)  05 {  06 dti<4>;  07 c(i<3>o<5>i<2>){  08 i<4>=i<3>;  09 }e{  10 i<4>=i<2>;  11 }  12 ri<4>;  13 }  15 m  16 {  17 dti<6>;  18 dti<7>;  19 i<7>=l<8>;  21 kti<9>=l<10>+i<7>-l<11>;  23 i<7>=l<12>-l<13>+b(l<15>,l<16>);  25 dti<17>;  26 i<17>=l<18>;  28 p(l<20>);  29 y(l<21>)  30 {  31 p(i<17>);  32 i<17>=i<17>+l<23>;  33 }  36 c(i<17>o<24>i<7>){  37 p(l<26>);  38 }e{  39 p(l<28>);  40 }  42 c(i<7>o<29>i<17>){  43 p(l<31>);  44 i<6>=i<1>(i<7>,i<17>);  45 p(i<6>);  46 }  49 kti<33>=l<34>;  50 p(i<33>);  53 dti<36>;  54 dti<37>;  56 p(l<39>);  57 i<36>=l<40>;  58 i<37>=l<41>;  60 dti<42>;  61 i<42>=a(i<36>,i<37>);  63 p(i<36>);  64 c(i<42>o<45>l<46>){  65 p(l<48>);  66 }e{  67 p(l<50>);  68 }  69 p(i<37>);  71 dti<52>;  72 i<52>=l<53>;  74 c(i<52>o<54>l<55>){  75 p(l<57>);  76 }  78 rl<58>;  79 } |

Листинг Б.1 – Таблица лексем

|  |
| --- |
| № > Идентификатор > Область видимости > Тип данных > Тип идентификатора > Индекс в ТЛ > Значение  0 > L1 > L1 > string > литерал > 1 > [43]"D:/LABS2/BPA-2023/BPA-lib/Debug/BPA-lib.lib"  1 > max > max > number > функция > 4 > нет значения  2 > a > maxa > number > параметр > 7 > нет значения  3 > b > maxb > number > параметр > 10 > нет значения  4 > res > maxres > number > переменная > 15 > 0  5 > > > maxres > оператор > 20 > нет значения  6 > d > maind > number > переменная > 44 > 0  7 > a > maina > number > переменная > 48 > 0  8 > L2 > L2 > number > литерал > 52 > 10  9 > cos > maincos > number > константа > 56 > нет значения  10 > L3 > L3 > number > литерал > 58 > 5  11 > L4 > L4 > number > литерал > 62 > 3  12 > L5 > L5 > number > литерал > 66 > 1000  13 > L6 > L6 > number > литерал > 68 > 100  14 > pow > mainpow > number > unknown > 70 > нет значения  15 > L7 > L7 > number > литерал > 72 > 2  16 > L8 > L8 > number > литерал > 74 > 2  17 > b > mainb > number > переменная > 79 > 0  18 > L9 > L9 > number > литерал > 83 > 1  19 > write > write > number > unknown > 85 > нет значения  20 > L10 > L10 > string > литерал > 87 > [11]"Count to 10"  21 > L11 > L11 > number > литерал > 92 > 10  22 > write > write > number > unknown > 95 > нет значения  23 > L12 > L12 > number > литерал > 104 > 1  24 > >= > L12 > оператор > 110 > нет значения  25 > write > write > number > unknown > 114 > нет значения  26 > L13 > L13 > string > литерал > 116 > [25]"b is greater with value: "  27 > write > write > string > unknown > 122 > нет значения  28 > L14 > L14 > string > литерал > 124 > [25]"a is greater with value: "  29 > != > L14 > оператор > 131 > нет значения  30 > write > write > string > unknown > 135 > нет значения  31 > L15 > L15 > string > литерал > 137 > [20]"max of a and b are: "  32 > write > write > string > unknown > 149 > нет значения  33 > lettera > mainlettera > string > константа > 157 > нет значения  34 > L16 > L16 > string > литерал > 159 > [1]"a"  35 > write > write > string > unknown > 161 > нет значения  36 > str3 > mainstr3 > string > переменная > 168 > [0]""  37 > str4 > mainstr4 > string > переменная > 172 > [0]""  38 > write > write > string > unknown > 174 > нет значения  39 > L17 > L17 > string > литерал > 176 > [46]"lexicographic string comparison str3 and str4:"  40 > L18 > L18 > string > литерал > 181 > [5]"hello"  41 > L19 > L19 > string > литерал > 185 > [6]"hello "  42 > res > mainres > number > переменная > 189 > 0  43 > compare > maincompare > number > unknown > 193 > нет значения  44 > write > write > number > unknown > 200 > нет значения  45 > == > write > оператор > 208 > нет значения  46 > L20 > L20 > number > литерал > 209 > 1  47 > write > write > number > unknown > 212 > нет значения  48 > L21 > L21 > string > литерал > 214 > [4]" == "  49 > write > write > string > unknown > 220 > нет значения  50 > L22 > L22 > string > литерал > 222 > [4]" != "  51 > write > write > string > unknown > 226 > нет значения  52 > info > maininfo > bool > переменная > 233 > нет значения  53 > L23 > L23 > bool > литерал > 237 > нет значения  54 > == > L23 > оператор > 242 > нет значения  55 > L24 > L24 > bool > литерал > 243 > нет значения  56 > write > write > bool > unknown > 246 > нет значения  57 > L25 > L25 > string > литерал > 248 > [4]"true"  58 > L26 > L26 > number > литерал > 253 > 0 |

Листинг Б.1 – Таблица идентификаторов

# **Приложение В**

Основные структуры таблиц лексем и идентификаторов данных

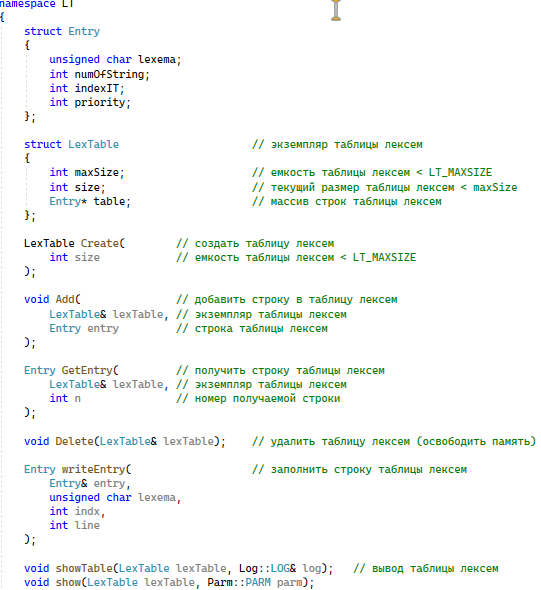


Рисунок В.1 – структура таблицы лексем

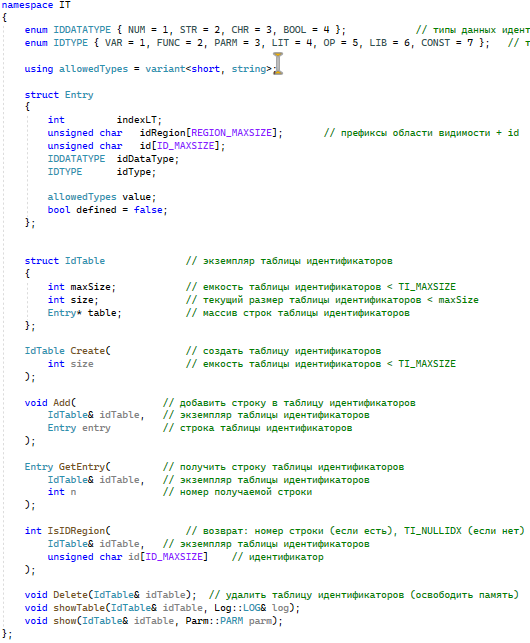


Рисунок В.2 – структура таблицы идентификаторов

# **Приложение Г**

Грамматика языка

|  |
| --- |
| Greibach greibach(  NS('S'), // стартовый символ  TS('$'), //дно стека  6,  Rule(  NS('S'),  GRB\_ERROR\_SERIES + 0, // неверная структура программы  5,  Rule::Chain(2, TS('u'), TS('l')),  Rule::Chain(3, TS('u'), TS('l'), NS('S')),  Rule::Chain(7, TS('m'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}')),  Rule::Chain(13, TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), NS('S')),  Rule::Chain(12, TS('t'), TS('f'), TS('i'), TS('('), TS(')'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), NS('S'))  ),  Rule(  NS('N'),  GRB\_ERROR\_SERIES + 1, // конструкции в функциях  18,  Rule::Chain(5, TS('d'), TS('t'), TS('i'), TS(';'), NS('N')),  Rule::Chain(4, TS('i'), TS('='), NS('E'), TS(';')),  Rule::Chain(5, TS('i'), TS('='), NS('E'), TS(';'), NS('N')),  Rule::Chain(7, TS('k'), TS('t'), TS('i'), TS('='), NS('E'), TS(';'), NS('N')),  Rule::Chain(6, TS('p'), TS('('), TS('i'), TS(')'), TS(';'), NS('N')),  Rule::Chain(6, TS('p'), TS('('), TS('l'), TS(')'), TS(';'), NS('N')),  Rule::Chain(12, TS('c'), TS('('), NS('C'), TS(')'), TS('{'), NS('N'), TS('}'), TS('e'), TS('{'), NS('N'), TS('}'), NS('N')),  Rule::Chain(8, TS('c'), TS('('), NS('C'), TS(')'), TS('{'), NS('N'), TS('}'), NS('N')),  Rule::Chain(4, TS('d'), TS('t'), TS('i'), TS(';')),  Rule::Chain(5, TS('p'), TS('('), TS('i'), TS(')'), TS(';')),  Rule::Chain(5, TS('p'), TS('('), TS('l'), TS(')'), TS(';')),  Rule::Chain(6, TS('k'), TS('t'), TS('i'), TS('='), NS('E'), TS(';')),  Rule::Chain(11, TS('c'), TS('('), NS('C'), TS(')'), TS('{'), NS('N'), TS('}'), TS('e'), TS('{'), NS('N'), TS('}')),  Rule::Chain(7, TS('c'), TS('('), NS('C'), TS(')'), TS('{'), NS('N'), TS('}')),  Rule::Chain(8, TS('y'), TS('('), TS('i'), TS(')'), TS('{'), NS('N'), TS('}'), NS('N')),  Rule::Chain(8, TS('y'), TS('('), TS('l'), TS(')'), TS('{'), NS('N'), TS('}'), NS('N')),  Rule::Chain(7, TS('y'), TS('('), TS('i'), TS(')'), TS('{'), NS('N'), TS('}')),  Rule::Chain(7, TS('y'), TS('('), TS('l'), TS(')'), TS('{'), NS('N'), TS('}'))  ),  Rule(  NS('E'),  GRB\_ERROR\_SERIES + 2, // ошибка в выражении  15,  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l')),  Rule::Chain(4, FUNC\_CALL),  Rule::Chain(4, POW\_CALL),  Rule::Chain(4, COMPARE\_CALL),  Rule::Chain(6, FUNC\_CALL, TS('+'), NS('E')),  Rule::Chain(6, POW\_CALL, TS('+'), NS('E')),  Rule::Chain(6, COMPARE\_CALL, TS('+'), NS('E')),  Rule::Chain(6, FUNC\_CALL, TS('-'), NS('E')),  Rule::Chain(6, POW\_CALL, TS('-'), NS('E')),  Rule::Chain(6, COMPARE\_CALL, TS('-'), NS('E')),  Rule::Chain(3, TS('i'), TS('+'), NS('E')),  Rule::Chain(3, TS('l'), TS('+'), NS('E')),  Rule::Chain(3, TS('i'), TS('-'), NS('E')),  Rule::Chain(3, TS('l'), TS('-'), NS('E'))  ),  Rule(  NS('F'),  GRB\_ERROR\_SERIES + 3, // ошибка в параметрах функции  2,  Rule::Chain(2, TS('t'), TS('i')),  Rule::Chain(4, TS('t'), TS('i'), TS(','), NS('F'))  ),  Rule(  NS('W'),  GRB\_ERROR\_SERIES + 4, // ошибка в параметрах вызываемой функции  4,  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l')),  Rule::Chain(3, TS('i'), TS(','), NS('W')),  Rule::Chain(3, TS('l'), TS(','), NS('W'))  ),  Rule(  NS('C'),  GRB\_ERROR\_SERIES + 5, // условие if  4,  Rule::Chain(3, TS('i'), TS('o'), TS('i')),  Rule::Chain(3, TS('i'), TS('o'), TS('l')),  Rule::Chain(3, TS('l'), TS('o'), TS('i')),  Rule::Chain(3, TS('l'), TS('o'), TS('l'))  )  ); |

Листинг Г.1 – Правила грамматики языка

# **Приложение Д**

Результат синтаксического анализа

|  |
| --- |
| Шаг : Правило Входная лента Стек  0 : S->ul ultfi(ti,ti){dti;c(ioi){i S$  0 : SAVESTATE: 1  0 : ultfi(ti,ti){dti;c(ioi){i ul$  1 : ltfi(ti,ti){dti;c(ioi){i= l$  2 : tfi(ti,ti){dti;c(ioi){i=i $  3 : TS\_NOK/NS\_NORULECHAIN  3 : RESSTATE  3 : ultfi(ti,ti){dti;c(ioi){i S$  4 : S->ulS ultfi(ti,ti){dti;c(ioi){i S$  4 : SAVESTATE: 1  4 : ultfi(ti,ti){dti;c(ioi){i ulS$  5 : ltfi(ti,ti){dti;c(ioi){i= lS$  6 : tfi(ti,ti){dti;c(ioi){i=i S$  7 : S->tfi(F){NrE;}S tfi(ti,ti){dti;c(ioi){i=i S$  7 : SAVESTATE: 2  7 : tfi(ti,ti){dti;c(ioi){i=i tfi(F){NrE;}S$  8 : fi(ti,ti){dti;c(ioi){i=i; fi(F){NrE;}S$  9 : i(ti,ti){dti;c(ioi){i=i;} i(F){NrE;}S$  10 : (ti,ti){dti;c(ioi){i=i;}e (F){NrE;}S$  11 : ti,ti){dti;c(ioi){i=i;}e{ F){NrE;}S$  12 : F->ti ti,ti){dti;c(ioi){i=i;}e{ F){NrE;}S$  12 : SAVESTATE: 3  12 : ti,ti){dti;c(ioi){i=i;}e{ ti){NrE;}S$  13 : i,ti){dti;c(ioi){i=i;}e{i i){NrE;}S$  14 : ,ti){dti;c(ioi){i=i;}e{i= ){NrE;}S$  15 : TS\_NOK/NS\_NORULECHAIN  …  1247: p(l);}rl;} N}rE;}$  1248: N->p(l); p(l);}rl;} N}rE;}$  1248: SAVESTATE: 75  1248: p(l);}rl;} p(l);}rE;}$  1249: (l);}rl;} (l);}rE;}$  1250: l);}rl;} l);}rE;}$  1251: );}rl;} );}rE;}$  1252: ;}rl;} ;}rE;}$  1253: }rl;} }rE;}$  1254: rl;} rE;}$  1255: l;} E;}$  1256: E->l l;} E;}$  1256: SAVESTATE: 76  1256: l;} l;}$  1257: ;} ;}$  1258: } }$  1259: $  1260: LENTA\_END  1261: ------>LENTA\_END |

Листинг Д.1 – результат синтаксического анализа

# **Приложение Е**

Сгенерированный код ассемблера

|  |
| --- |
| .586  .model flat, stdcall  includelib libucrt.lib  includelib kernel32.lib  includelib D:/LABS2/BPA-2023/BPA-lib/Debug/BPA-lib.lib  EXTERN \_printS :PROC  EXTERN \_printN :PROC  EXTERN \_pow :PROC  EXTERN \_compare :PROC  EXTERN \_pause :PROC  ExitProcess PROTO :DWORD  .stack 4096  .const  L1 BYTE 'D:/LABS2/BPA-2023/BPA-lib/Debug/BPA-lib.lib', 0  L2 WORD 10  maincos WORD 12  L3 WORD 5  L4 WORD 3  L5 WORD 1000  L6 WORD 100  L7 WORD 2  L8 WORD 2  L9 WORD 1  L10 BYTE 'Count to 10', 0  L11 WORD 10  L12 WORD 1  L13 BYTE 'b is greater with value: ', 0  L14 BYTE 'a is greater with value: ', 0  L15 BYTE 'max of a and b are: ', 0  mainlettera BYTE 'a', 0  L16 BYTE 'a', 0  L17 BYTE 'lexicographic string comparison str3 and str4:', 0  L18 BYTE 'hello', 0  L19 BYTE 'hello ', 0  L20 WORD 1  L21 BYTE ' == ', 0  L22 BYTE ' != ', 0  L23 WORD 1  L24 WORD 1  L25 BYTE 'true', 0  L26 WORD 0  .data  maxres WORD 0  maind WORD 0  maina WORD 0  mainb WORD 0  mainstr3 DWORD 0  mainstr4 DWORD 0  mainres WORD 0  maininfo WORD 0  .code  max PROC maxa : WORD, maxb : WORD  mov ax, maxb  cmp ax, maxa  jg p0  jl p1  je p1  p0:  push maxb  pop maxres  jmp ife0  p1:  push maxa  pop maxres  ife0:  push maxres  pop eax  ret  max ENDP  main PROC  push L2  pop maina  push L5  pop ebx  push L6  pop eax  sub ebx, eax  push ebx  pop ebx  push offset L8  push offset L7  call \_pow  push eax  add ebx, eax  push ebx  pop maina  push L9  pop mainb  mov ebx, ecx  push offset L10  call \_printS  mov ecx, ebx  movzx ecx, L11  p2:  mov ebx, ecx  push mainb  call \_printN  mov ecx, ebx  push mainb  pop ebx  push L12  pop eax  add ebx, eax  push ebx  pop mainb  loop p2  mov ax, mainb  cmp ax, maina  je p3  jg p3  jl p4  p3:  mov ebx, ecx  push offset L13  call \_printS  mov ecx, ebx  jmp ife1  p4:  mov ebx, ecx  push offset L14  call \_printS  mov ecx, ebx  ife1:  mov ax, maina  cmp ax, mainb  jl p5  jg p5  je p6  p5:  mov ebx, ecx  push offset L15  call \_printS  mov ecx, ebx  push mainb  push maina  call max  push eax  pop maind  mov ebx, ecx  push maind  call \_printN  mov ecx, ebx  p6:  mov ebx, ecx  push offset mainlettera  call \_printS  mov ecx, ebx  mov ebx, ecx  push offset L17  call \_printS  mov ecx, ebx  push offset L18  pop mainstr3  push offset L19  pop mainstr4  push offset mainstr4  push offset mainstr3  call \_compare  push eax  pop mainres  mov ebx, ecx  push mainstr3  call \_printS  mov ecx, ebx  mov ax, mainres  cmp ax, L20  je p7  jg p8  jl p8  p7:  mov ebx, ecx  push offset L21  call \_printS  mov ecx, ebx  jmp ife2  p8:  mov ebx, ecx  push offset L22  call \_printS  mov ecx, ebx  ife2:  mov ebx, ecx  push mainstr4  call \_printS  mov ecx, ebx  push L23  pop maininfo  mov ax, maininfo  cmp ax, L24  je p9  jg p10  jl p10  p9:  mov ebx, ecx  push offset L25  call \_printS  mov ecx, ebx  p10:  push 0  call \_pause  call ExitProcess  main ENDP  end main |

Листинг Е.1 – код на языке ассемблера