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 LMM-2023»

Выполнил студент Ляшонок Матвей Михайлович

(Ф.И.О.)

Руководитель проекта acc. Север Александра Сергеевна

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

Заведующий кафедрой к.т.н. Смелов Владимир Владиславович

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

Консультант acc. Север Александра Сергеевна

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

Курсовой проект защищен с оценкой

Минск 2023

Оглавление

[Введение 4](#_Toc122342056)

[1 Спецификация языка программирования 5](#_Toc122342057)

[1.1 Характеристика языка программирования 5](#_Toc122342058)

[1.2 Определение алфавита языка программирования 5](#_Toc122342059)

[1.3 Применяемые сепараторы 6](#_Toc122342060)

[1.4 Применяемые кодировки 6](#_Toc122342061)

[1.5 Типы данных 6](#_Toc122342062)

[1.6 Преобразование типов данных 7](#_Toc122342063)

[1.7 Идентификаторы 7](#_Toc122342064)

[1.8 Литералы 7](#_Toc122342065)

[1.9 Объявление данных 8](#_Toc122342066)

[1.10 Инициализация данных 8](#_Toc122342067)

[1.11 Инструкции языка 8](#_Toc122342068)

[1.12 Операции языка 9](#_Toc122342069)

[1.13 Выражения и их вычисление 10](#_Toc122342070)

[1.14 Конструкции языка 10](#_Toc122342071)

[1.15 Области видимости идентификаторов 11](#_Toc122342072)

[1.16 Семантические проверки 11](#_Toc122342073)

[1.17 Распределение оперативной памяти на этапе выполнения 11](#_Toc122342074)

[1.18 Стандартная библиотека и её состав 11](#_Toc122342075)

[1.19 Ввод и вывод данных 12](#_Toc122342076)

[1.20 Точка входа 12](#_Toc122342077)

[1.21 Препроцессор 12](#_Toc122342078)

[1.22 Соглашения о вызовах 12](#_Toc122342079)

[1.23 Объектный код 13](#_Toc122342080)

[1.24 Классификация сообщений транслятора 13](#_Toc122342081)

[1.25 Контрольный пример 13](#_Toc122342082)

[Приложение А 40](#_Toc122342132)

# Введение

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

Задание на курсовой проект можно разделить на следующие задачи:

1) Разработка спецификации языка LMM-2023. Для выполнения данной задачи необходимо детально определить лексические единицы - идентификаторы, ключевые слова, оператор присваивания, логические и сравнительные операторы; подробно описать синтаксис – структуру программы на языке, объявление переменных, определение функций, операторы присваивания, ветвления, вывода; изложить спецификацию типов данных – беззнаковые целочисленные; занимаемый объём памяти для каждого типа; правила именования идентификаторов - допустимые символы, регистр, длина имен.

2) Разработка лексического анализатора. С учётом поставленной задачи необходимо разработать алгоритм разбиения исходного кода на лексемы с учетом разделителей, многосимвольных лексем; реализовать распознавание идентификаторов, ключевых слов, обработка ошибок; создать таблицы лексем с их атрибутами (тип, значение, номер строки); механизм выдачи ошибок лексического анализа с указанием номера строк.

3)Разработка синтаксического анализатора. Для достижения целей необходимо рассмотреть алгоритм сопоставления входной последовательности лексем с синтаксическими правилами языка; построение дерева разбора, отражающего иерархическую структуру программы; проверку типов и согласованности в выражениях, присваиваниях, при вызове функций; механизм выдачи ошибок синтаксического анализа.

4) Разработка семантического анализатора. В данной задаче важны: проверка объявлений идентификаторов; анализ присваиваний и вызовов функций, построение графа потока управления, выявление неиспользуемых переменных, выдача предупреждений о потенциальных ошибках;

5)Разбор арифметических выражений. В данной ситуации требуется реализовать алгоритм разбора с учетом приоритетов операций, проверка типов операндов, вычисление значений констант, выдача ошибок и предупреждений, оптимизация выражений.

6)Разработка генератора кода. В контексте задачи важно учесть оптимизацию внутреннего представления программы, распределение регистров.

7)Тестирование транслятора. Для достижения целей стоит протестировать разработанные тестовые программы, выявить и исправить ошибки; использовать регрессионное тестирование; протестировать скомпилированные программы.

**1 Спецификация языка программирования**

**1.1 Характеристика языка программирования**

Язык программирования LMM-2023 является статическим сильно типизированным, процедурным, компилируемым.

**1.2 Определение алфавита языка программирования**

Алфавит языка программирования LMM-2023 использует таблицу символом Windows-1251, представленную ниже на рис 1.1.



Рисунок 1.1 Алфавит входных символов

В исходном коде на языке LMM-2023 допускается использование только строчных латинских букв, цифр от 0 до 9 и символов кириллицы, которые можно употреблять исключительно внутри строковых литералов.

**1.3 Применение сепараторов**

Специальные символы-разделители используются для разбиения исходного кода программы на лексемы (токены) в процессе лексического анализа. Эти символы-сепараторы выступают в роли границ между отдельными лексическими единицами языка. Список таких разделительных символов приведен в таблице 1.1.

Таблица 1.1 Сепараторы

|  |  |
| --- | --- |
| Символ(ы) | Назначение |
| «пробел» | Разделитель цепочек. Допускается везде кроме названий идентификаторов и ключевых слов |
| {…} | Блок функции или условной конструкции |
| (…) | Блок фактических или формальных параметров функции, а также приоритет арифметических операций |
| , | Разделитель параметров функций |
| & | ^ | Логические операции |
| ; | Разделитель программных конструкций |
| = | Оператор присваивания |
| >= <= < > == != | Операторы сравнения |

Они позволяют отделять идентификаторы, ключевые слова, константы и прочие лексемы друг от друга при разборе исходного кода, чтобы корректно выполнить лексический анализ.

**1.4 Применяемые кодировки**

В языке программирования для записи исходного кода программ применяется кодировка Windows-1251. Она включает в себя как латинский, так и кириллический алфавиты, а также набор специальных символов. Среди них скобки [ ], круглые скобки ( ), запятая ,, точка с запятой ;, двоеточие :, решетка #, знаки арифметических операций +, -, /, \\*, знаки сравнения >, <, амперсанд &, восклицательный знак !, а также фигурные скобки {}.

**1.5 Типы данных**

**Написать текст, чтобы таблица быфла на другой странце**

Таблица 1.2 Типы данных языка LMM-2023

|  |  |
| --- | --- |
| Тип данных | Характеристики |
| Беззнаковый целочисленный тип данных **PosInt** | Фундаментальный тип данных. Используется для работы с числовыми значениями от 0 до 4 294 967 295. При переполнении данного типа данных будет выведена ошибка. В памяти занимает 4 байта.  Поддерживаемые операции:  = (бинарный) – оператор присваивания;  & (бинарный) – результат истинен, если оба операнда истинны;  | (бинарный) – результат истинен, если хотя бы один операнд истинен;  ^ (унарный) – операция инвертирования числа на противоположное.  В качестве условия условного оператора поддерживаются следующие логические операции:  < (бинарный) – оператор «меньше»;  > (бинарный) – оператор «больше»;  != (бинарный) – оператор неравенства;  == (бинарный) – оператор равенства; |
| Символьный тип данных **Litara** | Фундаментальный тип данных. Используется для работы с символом. По умолчанию инициализируется нулевым символом \0. Занимает 1 байт. Максимальное количество символов – 1. При переполнении данного типа данных, переменная будет инициализироваться первым символом.  Операции над данными строкового типа: присваивание строковому идентификатору значения другого строкового идентификатора, строкового литерала или значения строковой функции, а также использование библиотечных функций. |

Язык LMM-2023 позволит работать с беззнаковыми целочисленными значениями и символами, а также проводить операции над ними.

**1.6 Преобразование типов данных**

Явное преобразование типов данных предусмотрено функцией стандартной библиотеки: преобразование строки в число. Больше видов преобразования не предусмотрено.

**1.7 Идентификаторы**

Общее количество идентификаторов ограничено размером таблицы идентификаторов, максимальная длина которой равна пяти символам. Идентификаторы должны содержать только символы нижнего регистра латинского алфавита. Если идентификатор объявлен внутри функционального блока, он будет иметь префикс, который совпадает с именем функции, в пределах которой он объявлен. Этот префикс занимает 5 дополнительных символов. В случае, если длина идентификатора превышает 10 символов (5 символов для имени и 5 символов для префикса), он будет усечен до этой максимальной длины. Эти правила применяются ко всем видам идентификаторов, включая имена переменных, функций и параметров функций. Зарезервированные идентификаторы отсутствуют. Кроме того, идентификаторы не могут совпадать с ключевыми словами. Типы идентификаторов включают имена переменных, имена функций и имена параметров функций. Имена функциональных идентификаторов и переменных не должны совпадать с именами команд ассемблера.

Пример правильного идентификатора: *numA*, *numB*. Пример неправильного идентификатора: *PosInt* (совпадает с зарезервированным словом).

**1.8 Литералы**

Переменные инициализируются с использованием литералов, которые всегда являются *rvalue*. Типы доступных литералов в языке LMM-2023 описаны в таблице 1.3.

Таблица 1.3 Литералы

|  |  |
| --- | --- |
| Литералы | Пояснение |
| Беззнаковые целочисленные литералы в десятичном представлении | Может состоять из чисел [0-9]. Минимальное значение равно 0, максимальное 4 294 967 295. При выходе за пределы будет выведена ошибка. |
| Символьные литералы | Один символ, заключённый в одинарные кавычки. |
| Логический литерал в произвольном представлении | Представляет собой ключевые слова “FALSE” “TRUE”, которые являются 0 и 1 соответственно. |

Правила для строковых литералов в языке GKV-2022 запрещают вставку одинарных и двойных кавычек внутри самих строковых литералов.

**1.9 Объявление переменных**

При объявлении переменной указываются ключевое слово *var* тип данных и имя идентификатора, и при этом также разрешается провести инициализацию переменной.

Пример объявления числового типа с инициализацией:

*var PosInt val = 24;*

*var PosInt rVal = true.*

Пример объявления переменной символьного типа с инициализацией:

*var Litara char = ‘Z’.*

Для декларации функций и процедур важно использовать ключевое слово “function”, предварительно указав тип функции. Затем следует обязательное перечисление параметров и определение тела функции.

**1.10 Инициализация данных**

При объявлении переменной можно провести начальную инициализацию данных. В этом случае, переменной будет присвоено значение литерала или идентификатора, указанного справа от символа присвоения. При этом объектами-инициализаторами могут быть только идентификаторы или литералы. Если переменная объявляется без явной инициализации, то для типа "PosInt" по умолчанию устанавливается значение 0, для типа Litara – “\0”.

**1.11 Инструкции языка**

Инструкции языка LMM-2023 представлены в таблице 1.4.

|  |  |
| --- | --- |
| Инструкция | Реализация |
| Объявление переменной | var <тип данных> <идентификатор>; |
| Объявление переменной с явной инициализацией | <тип данных> <идентификатор> = <значение>;  Значение – инициализатор конкретного типа. Может быть только литералом или идентификатором |
| Возврат из функции или процедуры | Для функций, возвращающих значение:  **back** <идентификатор/литерал>;  Для процедур:  **back;** |
| Вывод данных | **write**<идентификатор/литерал>; |
| Вызов функции или процедуры | <идентификатор функции> (<список параметров>);  Список параметров может быть пустым. |
| Присваивание  Инструкция | <идентификатор> **=** <выражение>;  Выражением может быть идентификатор, литерал, или вызов функции соответствующего типа. Для беззнакового целочисленного типа выражение может быть дополнено логическими операциями с любым количеством операндов с использованием скобок. Для символьного Реализация |
|  | типа выражение может быть только идентификатором, литералом или вызовом функции, возвращающей значение символьного типа. |

## 1.12 Операции языка

В языке LMM-2023 определены следующие операции для работы с данными. Операции умножения имеют более высокий приоритет по сравнению с операциями сложения и вычитания. Для установки наивысшего приоритета операций следует использовать круглые скобки. Таблица 1.5 содержит подробное описание доступных операций в языке.

Таблица 1.5 Операции языка LMM-2023

|  |  |
| --- | --- |
| Тип оператора | Оператор |
| Логические | 1. & – логическое “и” 2. | – логическое “или” 3. ^ – инверсия |
| Присвоения | 1. = –присвоение |
| Сравнительные | 1. **>** – больше  2. **<** – меньше  3. == – равенство  4. != – неравенство  5. <= – меньше или равно  6. >= – больше или равно |

# 1.13 Выражения и их вычисления

Вычисление выражений представляет собой одну из ключевых задач в языках программирования. Построение любого выражения подчиняется следующим правилам:

1Для изменения приоритета операций можно использовать скобки.

1.Выражение должно быть записано в одну строку, без переносов.

2.Недопустимо использование двух операторов, следующих друг за другом.

3.В рамках выражения допускается включение вызовов функций, возвращающих беззнаковое целочисленное значение.

Перед генерацией кода каждое выражение преобразуется в обратную польскую запись для упрощения последующего вычисления выражения на языке ассемблера.

**1.14 Конструкции языка**

Программа на языке LMM-2023 оформляется в виде функций пользователя и главной функции. При составлении функций рекомендуется выделять блоки и фрагменты и применять отступы для лучшей читаемости кода, представленные в таблице 1.6.

Таблица 1.6 Программные конструкции языка LMM-2023

|  |  |  |
| --- | --- | --- |
| Конструкция | | Реализация |
| Главная функция | | **glavnaya**  **{**…  **}** |
| Внешняя функция | | <тип данных> **function** <идентификатор> **(**<тип> <идентификатор>, ...**)**  **{**…  **back** <идентификатор/литерал>;  **}** |
| Внешняя процедура | **procedure** <идентификатор> **(** <тип> <идентификатор>, ...**)**  **{**…  **}** | |
| Условная конструкция | **if:** <идентификатор1> <оператор> <идентификатор2>  **блок1{**…**}**  **else блок2{**…**}**  <идентификатор1>, <идентификатор2> - идентификаторы или литералы беззнакового целочисленного типа (но не два литерала одновременно). <оператор> - один из операторов сравнения (**> < != ==**), устанавливающий отношение между двумя операндами и организующий условие данной конструкции. При истинности условия выполняется код внутри блока1, иначе – код внутри блока **блок2**. | |

# 1.15 Область видимости идентификаторов

Область видимости в языке LMM-2023 организована так же, как в С++, сверху вниз. Это означает, что переменные, объявленные в одной функции, недоступны в других. Все объявления переменных и операции с ними выполняются внутри блоков. Каждая переменная или параметр функции получает префикс, соответствующий имени функции, внутри которой они были объявлены. Все идентификаторы считаются локальными и должны быть объявлены внутри какой-либо функции. Глобальных переменных не существует, и параметры видны только внутри функции, в которой они были объявлены.

## 1.16 Семантические проверки

В языке программирования LMM-2023 выполняются следующие семантические проверки:

1. Наличие функции **glavnaya** – 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 символьных выражений;
8. Превышение размера строковых и числовых литералов;
9. Правильность составленного условия условного оператора.

## 1.17 Распределение оперативной памяти на этапе выполнения

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

## 1.18 Стандартная библиотека и её состав

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

Таблица 1.8 Стандартная библиотека языка LMM-2023

|  |  |
| --- | --- |
| Функция | Описание |
| PosInt toPosInt(Litara **s**); | Преобразование символа в беззнаковое целое число. |
| Litara toLit(PosInt **a**); | Преобразование беззнакового целого числа в символ |

Стандартная библиотека написана на языке C++ и интегрируется с кодом на этапе генерации. Вызовы стандартных функций могут быть осуществлены там, где доступны вызовы пользовательских функций. В стандартной библиотеке также реализованы функции для управления выводом, но они недоступны для конечного пользователя. Для вывода данных предусмотрен оператор "write". Подробная информация о доступных функциях представлена в таблице 1.9.

Таблица 1.9 Дополнительные функции стандартной библиотеки

|  |  |
| --- | --- |
| Функция на языке С++ | Описание |
| void write(int value) | Функция для вывода в стандартный поток значения беззнакового целочисленного идентификатора/литерала. |
| void write(char\* value) | Функция для вывода в стандартный поток значения символьного идентификатора/литерала. |

## 1.19 Ввод и вывод данных

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

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

## 1.20 Точка входа

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

## 1.21 Препроцессор

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

## 1.22 Соглашения о вызовах

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

– все параметры функции передаются через стек;

– память высвобождает вызываемый код;

– занесение в стек параметров идёт справа налево.

## 1.23 Объектный код

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

## 1.24 Классификация сообщений транслятора

Сообщения, которые генерирует транслятор, влияют на степень информативности транслятора. Следовательно, сообщения от транслятора должны предоставлять наиболее полную информацию о возможных ошибках, допущенных пользователем при написании программы. Вы можете найти список сообщений от транслятора в таблице 1.10.

Таблица 1.10 Классификация ошибок

|  |  |
| --- | --- |
| Номера ошибок | Характеристика |
| 0 – 200 | Системные ошибки |
| 200 – 299 | Ошибки лексического анализа |
| 300 – 399 | Ошибки семантического анализа |
| 600 – 699 | Ошибки синтаксического анализа |

## 1.25 Контрольный пример

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

# 2 Структура транслятора

## 2.1 Компоненты транслятора, их назначение и принципы взаимодействия

В языке LMM-2022 исходный код транслируется в язык Assembler. Транслятор языка разделён на отдельные части, которые взаимодействуют между собой и выполняют отведённые функции, представленные в пункте 2.1. Получение ассемблерного кода осуществляется с использованием выходных данных лексического анализатора – таблицы лексем и таблицы идентификаторов. Для указания выходных файлов применяются входные параметры транслятора, описанные в таблице 2.1. Структура транслятора языка LMM-2022 представлена на рисунке 1.



Рисунок 2.1 – Структура транслятора

Лексический анализ – первая фаза трансляции. Назначением лексического анализатора является нахождение ошибок лексики языка LMM-2023 и формирование таблицы лексем и таблицы идентификаторов.

Синтаксический анализ – это основная часть транслятора, предназначенная для распознавания синтаксических конструкций и формирования промежуточного кода LMM-2023. Для этого используются таблица лексем и идентификаторов. Синтаксический анализатор распознаёт синтаксические конструкции, выявляет синтаксические ошибки при их наличии и формирует дерево разбора

Семантический анализ в свою очередь является проверкой исходной программы LMM-2023 на семантическую согласованность с определением языка, т.е. проверяет правильность текста исходной программы с точки зрения семантики.

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 LMM-2023, прошедший успешно 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 LMM-2023

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<имя\_файла> | Входной файл с любым расширением, в котором содержится исходный код на языке LMM-2023. 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 LMM-2023. В этот файл могут быть выведены таблицы идентификаторов, лексем, а также дерево разбора. | <имя\_файла>.log |
| -out:<имя\_файла> | В этот файл будет записан результат трансляции кода на язык assembler | <имя\_файла>.asm |

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

## 2.3 Перечень протоколов, формируемых транслятором и их содержимое

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

Таблица 2.2 – Протоколы, формируемые транслятором языка LMM-2023

|  |  |
| --- | --- |
| Формируемый протокол | Описание протокола |
| Файл журнала, log | Файл содержит в себе краткую информацию об исходном коде на языке LMM-2023. В этот файл выводится различные ошибки, а также результат работы анализаторов |
| Файл журнала, logOut | Файл содержит в себе краткую информацию об исходном коде на языке LMM-2023. В этот файл выводится протокол работы анализаторов. |
| asm | Содержит сгенерированный код на языке Ассемблера. |

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

# Глава 3. Разработка лексического анализатора

## 3.1 Структура лексического анализатора

Лексический анализатор – часть транслятора, выполняющая лексический анализ. Лексический анализатор принимает обработанный и разбитый на отдельные компоненты исходный код на языке LMM-2023. На выходе формируется таблица лексем и таблица идентификаторов. Структура лексического анализатора представлена на рисунке 3.1.



Рисунок 3.1 Структура лексического анализатора LMM-2023

## 3.2 Контроль входных символов

Таблица для контроля входных символов представлена на рисунке 3.2

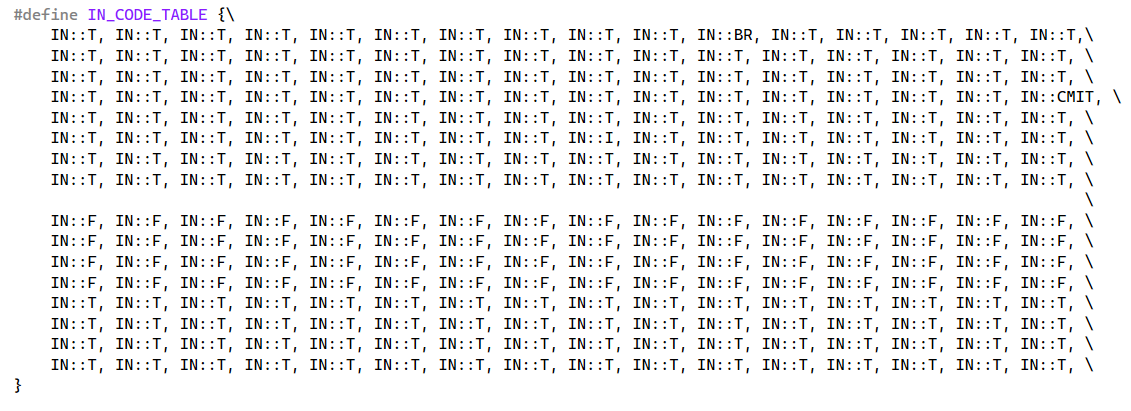


Рисунок 3.2. Таблица контроля входных символов

Принцип работы таблицы заключается в соответствии значения каждому элементу в шестнадцатеричной системе счисления значению в таблице WINDOWS-1251.

Описание значения символов: T – разрешённый символ, F – запрещённый символ, I – игнорируемый символ, BR – символ перехода на новую строку (\n), CMIT – символ комментария(?).

## 3.3 Удаление избыточных символов

Избыточными символами являются символы табуляции и пробелы.

Избыточные символы удаляются на этапе разбиения исходного кода на лексемы.

Описание алгоритма удаления избыточных символов:

1. Посимвольно считываем исходный код, занесенный в структуру In.

2. Встреча пробела или знака табуляции вне пределов строкового литерала является своего рода встречей символа-сепаратора.

3. В отличие от других символов-сепараторов не записываем в таблицу 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

|  |  |  |
| --- | --- | --- |
| Тип цепочки | Цепочка | Лексема |
| Ключевые слова | var | d |
| PosInt, string, boolean, Litara | t |
| glavnaya | m |
| method | f |
| procedure | p |
| back | r |
| write | o |
| cycle | u |
| if | w |
| else | ! |
| Иное | Идентификатор | i |
| Литерал | l |

Продолжение таблицы 3.1

|  |  |  |
| --- | --- | --- |
| Сепараторы | ; | ; |
| , | , |
| { | { |
| } | } |
| ( | ( |
| ) | ) |
| Операторы | Арифметические (+, -, \*, /, %), Cравнения (!,~,<,>) | v |
| Логические(^, |, &) | v |
| Присваивание (=) | = |

Пример реализации таблицы лексем представлен в приложении Б.

Также в приложении В находятся конечные автоматы, соответствующие лексемам языка LMM-2023.

## 3.5 Основные структуры данных

Описание основных структур данных, используемых для хранения таблиц идентификаторов, представлено на рис. 3.3.

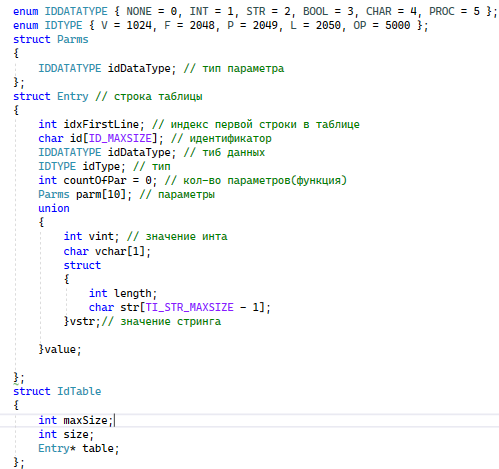


Рисунок 3.3. — Структуры таблиц идентификаторов LMM-2023

Описание основных структур данных, используемых для хранения таблиц лексем, представлено на рис. 3.4.

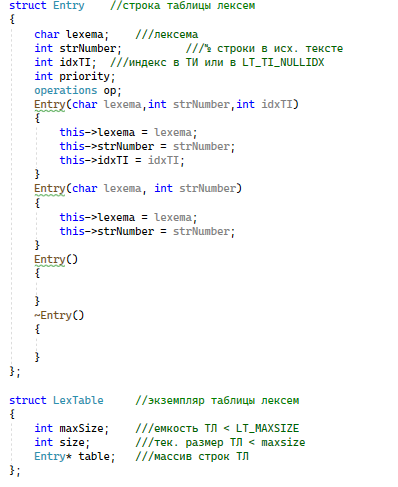


Рисунок 3.4 — Структуры таблиц лексем LMM-2023

## 3.6 Принцип обработки ошибок

Когда возникает ошибка – работа транслятора прекращается, а ошибка записывается в log журнал.

## 3.7 Структура и перечень сообщений лексического анализатора

Перечень сообщений лексического анализатора представлен на рисунке 3.5.

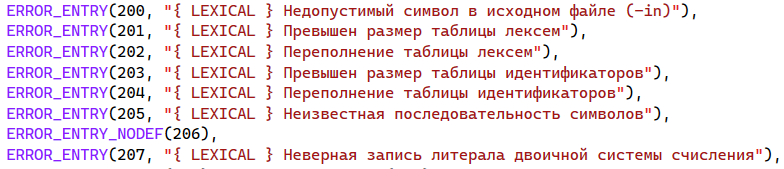


Рисунок 3.5 – Перечень ошибок лексического анализатора

## 3.8 Параметры лексического анализатора и режимы его работы

Входным параметром лексического анализатора является структура IN, которая содержит исходный текст программы, написанный на языке LMM-2023, а также структура LOG, которая содержит файл протокола.

## 3.9 Алгоритм лексического анализа

Лексический анализ выполняется программой (входящей в состав транслятора), называемой лексическим анализатором. Цель лексического анализа — выделение и классификация лексем в тексте исходной программы. Лексический анализатор производит распознаёт и разбирает цепочки исходного текста программы. Это основывается на работе конечных автоматов, которую можно представить в виде графов. Регулярные выражения — аналитический или формульный способ задания регулярных языков. Они состоят из констант и операторов, которые определяют множества строк и множество операций над ними. Любое регулярное выражение можно представить в виде графа. Для ускорения работы анализатора я добавил просмотр первого символа слова, за счет этого отсеиваются неподходящие графы. Результат работы лексического анализатора – сформированные таблицы лексем и идентификаторов.

Пример. Регулярное выражение для ключевого слова glavnaya.

Граф конечного автомата для этой лексемы представлен на рисунке 3.6. S0 – начальное состояние, S8 – конечное состояние автомата. В виде кода представлен на рисунке 3.7.



Рисунок 3.6 — Граф переходов для цепочки “primary”

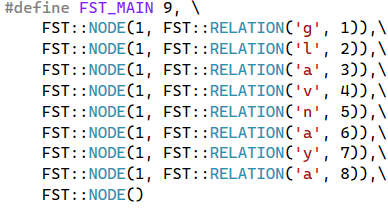


Рисунок 3.7 — Граф переходов для цепочки “ primary ”

## 3.10 Контрольный пример

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

**Приложение А**

Листинг 1 – Исходный код программы на языке LMM-2023

|  |
| --- |
| boolean method EvenCheck( PosInt k)  {  var boolean h;  k= k % 2;  if (k ~ 0)  {  h =TRUE;  }  else  {  h =FALSE;  }  back h;  }  PosInt method GetFact( PosInt n)  {  var PosInt k;  var PosInt j;  if (n < 0)  {  k = 0;  }    if (n ~ 0)  {  k = 1;  }  else  {  j=n-1;  k=n\*GetFact(j);  }  back k;  }  procedure Count( string h)  {  ? write "счет от нуля до длины строки: ";  write h;  hortab;  var PosInt length = strlength (h);  var PosInt i=0;  cycle (i < length)  {  write i;  write ',';  i=i+1;  }  }  glavnaya  {  var PosInt number = GetFact(7); ? вычисляем факториал  ? write "факториал числа равен: ";  write number;  hortab ;  var boolean iseven = EvenCheck(number); ? Проверка на четность числа  write iseven;  hortab ;  Count("hello");  hortab ;  var PosInt binary = ^10001010;  write binary;  hortab ;  var PosInt octal = 015;  var PosInt result = strlength ("hello")+((^1010+i012+45)/(2+(4\*5+1)/3))+ stoi ("12")+(octal+binary)%2;  write result;  hortab ;  var PosInt g = strcomp("hello","hell");  write g;  hortab ;  } |