МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора MTD-2020»

Выполнил студент Мядель Тимофей Дмитриевич

(Ф.И.О.)

Руководитель проекта ст.пр. Наркевич Аделина Сергеевна

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

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

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

Консультанты ст.пр. Наркевич Аделина Сергеевна

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

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

Нормоконтролер ст.пр. Наркевич Аделина Сергеевна

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

Курсовой проект защищен с оценкой

Минск 2020

**Содержание**

[Введение 2](#_Toc59102038)

[Глава 1 Спецификация языка 2](#_Toc59102039)

[1.1. Характеристика языка программирования 2](#_Toc59102040)

[1.2. Определение алфавита языка программирования 2](#_Toc59102041)

[1.3. Применяемые сепараторы 2](#_Toc59102042)

[1.4. Применяемые кодировки 2](#_Toc59102043)

[1.5. Типы данных 2](#_Toc59102044)

[1.6. Преобразование типов данных 2](#_Toc59102045)

[1.7. Идентификаторы 2](#_Toc59102046)

[1.8. Литералы 2](#_Toc59102047)

[1.9. Объявление данных 2](#_Toc59102048)

[1.10. Инициализация данных 2](#_Toc59102049)

[1.11. Инструкции языка 2](#_Toc59102050)

[1.12. Операции языка 2](#_Toc59102051)

[1.13. Выражения и их вычисление 2](#_Toc59102052)

[1.14. Конструкции языка 2](#_Toc59102053)

[1.15. Область видимости идентификаторов 2](#_Toc59102054)

[1.16. Семантические проверки 2](#_Toc59102055)

[1.17. Распределение оперативной памяти на этапе выполнения. 2](#_Toc59102056)

[1.18. Стандартная библиотека и ее состав 2](#_Toc59102057)

[1.19. Ввод и вывод данных 2](#_Toc59102058)

[1.20. Точка входа 2](#_Toc59102059)

[1.21. Препроцессор 2](#_Toc59102060)

[1.22. Соглашения о вызовах 2](#_Toc59102061)

[1.23. Объектный код 2](#_Toc59102062)

[1.24. Классификация сообщений транслятора 2](#_Toc59102063)

[1.25. Контрольный пример 2](#_Toc59102064)

[Глава 2 Структура транслятора 2](#_Toc59102065)

[2.1 Компоненты транслятора их назначение и принципы взаимодействия 2](#_Toc59102066)

[2.2 Перечень входных параметров транслятора 2](#_Toc59102067)

[2.3 Перечень протоколов, формируемых транслятором и их содержимое 2](#_Toc59102068)

[Глава 3 Разработка лексического анализатора 2](#_Toc59102069)

[3.1 Структура лексического анализатора 2](#_Toc59102070)

[3.2 Контроль входных символов 2](#_Toc59102071)

[3.3 Удаление избыточных символов 2](#_Toc59102072)

[3.4 Перечень ключевых слов, сепараторов, символов операций соответствующим им лексемам и конечных автоматов 2](#_Toc59102073)

[3.5 Основные структуры данных 2](#_Toc59102074)

[3.6 Принцип обработки ошибок 2](#_Toc59102075)

[3.7 Структура и перечень сообщений лексического анализатора 2](#_Toc59102076)

[3.8 Параметры лексического анализатора и режимы его работы 2](#_Toc59102077)

[3.9 Алгоритм лексического анализа 2](#_Toc59102078)

[3.10 Контрольный пример 2](#_Toc59102079)

[Глава 4 Разработка синтаксического анализатора 2](#_Toc59102080)

[4.1 Структура синтаксического анализатора 2](#_Toc59102081)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 2](#_Toc59102082)

[4.3 Построение конечного магазинного автомата 2](#_Toc59102083)

[4.4 Основные структуры данных 2](#_Toc59102084)

[4.5 Описание алгоритма синтаксического разбора 2](#_Toc59102085)

[4.6 Структура и перечень сообщений синтаксического анализатора 2](#_Toc59102086)

[4.7 Параметры синтаксического анализатора и режимы его работы 2](#_Toc59102087)

[4.8 Принцип обработки ошибок 2](#_Toc59102088)

[4.9 Контрольный пример 2](#_Toc59102089)

[Глава 5 Разработка семантического анализатора 2](#_Toc59102090)

[5.1 Структура семантического анализатора 2](#_Toc59102091)

[Функции семантического анализатора 2](#_Toc59102092)

[5.2 Структура и перечень сообщений семантического анализатора 2](#_Toc59102093)

[5.3 Принцип обработки ошибок 2](#_Toc59102094)

[Глава 6 Вычисление выражений 2](#_Toc59102095)

[6.1 Выражения, допускаемые языком 2](#_Toc59102096)

[6.2 Польская запись и принцип ее построения 2](#_Toc59102097)

[6.3 Программная реализация обработки выражений 2](#_Toc59102098)

[6.4 Контрольный пример 2](#_Toc59102099)

[Глава 7 Генерация кода 2](#_Toc59102100)

[7.1 Структура генератора кода 2](#_Toc59102101)

[7.2 Представление типов данных в оперативной памяти 2](#_Toc59102102)

[7.3 Статическая библиотека 2](#_Toc59102103)

[7.4 Особенности алгоритма генерации кода 2](#_Toc59102104)

[7.5 Входные параметры генератора кода 2](#_Toc59102105)

[7.6 Контрольный пример 2](#_Toc59102106)

[Глава 8 Тестирование транслятора 2](#_Toc59102107)

[8.1 Общие положения 2](#_Toc59102108)

[8.2 Результаты тестирования 2](#_Toc59102109)

[Заключение 2](#_Toc59102110)

[Приложение А 2](#_Toc59102111)

[Приложение Б 2](#_Toc59102112)

[Приложение Д 2](#_Toc59102113)

[Приложение Е 2](#_Toc59102114)

[Список использованных источников 2](#_Toc59102115)

**Введение**

Целью выполнения курсового проекта по дисциплине «Языки программирования» является приобретение навыков разработки системы программирования (трансляторов, интерпретаторов). В данном курсовом проекте поставлена задача разработки собственного языка программирования и транслятора для него. Название языка – MTD-2020. Написание транслятора будет осуществляться на языке C++, при этом код на языке MTD-2020 будет транслироваться в язык JavaScript.

Транслятор MTD-2020 состоит из следующих частей:

– лексический и семантический анализаторы;

– синтаксический анализатор;

– генератор исходного кода на языке JavaScript. Задание на курсовой проект можно разделить на следующие задачи:

– разработка спецификации языка MTD-2020;

– разработка лексического анализатора;

– разработка синтаксического анализатора;

– разработка семантического анализатора;

– разбор арифметических выражений;

– разработка генератора кода;

– тестирование транслятора.

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

# **Глава 1 Спецификация языка**

## **1.1. Характеристика языка программирования**

MTD-2020 – используется для школьников, для введения в специальность программирования. Язык программирования является компилируемым, строго типизированным, процедурным, высокоуровневым языком программирования. Поддерживающий процедурную парадигму программирования.

## **1.2. Определение алфавита языка программирования**

В языке MTD-2020 разрешено использовать латинские символы [A- Z, a- z], арабские цифры [0-9], а также сепараторы и непечатные символы пробела, перевода строки и табуляции. Символы русского языка разрешены только в строковых литералах.

## **1.3. Применяемые сепараторы**

Язык MTD-2020 разрешает использовать сепараторы, для написания исходного кода, представленные в таблице.

Символы-сепараторы – символы, используемые для разделения отдельных лексических единиц или функциональных элементов в исходном коде программы. Символы, которые являются сепараторами представлены в таблице 1.1.

Таблица 1.1 - Применяемые сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение сепаратора |
| .  “ “ (пробел)  , | Символы-сепараторы для разделения инструкций, параметров функции |
| [  ] | Символы-сепараторы указывающие на программный блок |
| (  ) | Символы-сепараторы указывающие на параметры/приоритетность операций(в выражениях) |

## **1.4. Применяемые кодировки**

В языке программирования MTD-2020 используется кодировка Windows-1251, представлена на рисунке 1.1.

Рисунок 1.1 — Основная таблица Windows-1251

## **1.5. Типы данных**

Предусмотрены три типа данных: беззнаковый целый (1 байт), строковый и логический, представленные в таблице 1.2.

Таблица 1.2 — Типы данных

|  |  |  |  |
| --- | --- | --- | --- |
| Название типа данных | Описание типа данных | Размерность | Область значений |
| uwhole | Беззнаковый целочисленный тип | 4 байта | От 0 до 255 |

Продолжение таблицы 1.2.

|  |  |  |  |
| --- | --- | --- | --- |
| logic | Логический тип | 1 байт | 0 или 1 |
| string | Строка(содержит любые символы, максимум 255) | От 0 до 255 байт (зависит от количества символов в строке | От 0 до 255 |

## **1.6. Преобразование типов данных**

В языке MTD-2020 преобразование не поддерживается.

## **1.7. Идентификаторы**

Идентификатор — имя компонента программы (переменной или функции), составленное программистом по определенным правилам.

В идентификаторах языка MTD-2020 разрешается использовать латинские буквы только нижнего регистров.

Идентификаторы не могут совпадать с ключевыми словами. Данные правила справедливы как для переменных, так и для функций.

Примеры ключевых слов: var, let, new, for и т.д.

## **1.8. Литералы**

Литерал — запись в исходном коде компьютерной программы, представляющая собой фиксированное значение. Литералами также называют представление значения некоторого типа данных.

В языке MTD-2020 существует 3 вида литералов: литералы целого типа, строковые и логические (true и false), описаны в таблице 1.3.

Таблица 1.3 – Литералы

|  |  |
| --- | --- |
| Тип | Описание |
| Литералы целого типа | Интерпретируются как uwhole, являются rvalue. Задаются в десятичной форме, либо восьмеричной. Должны быть неотрицательными. |
| Строковые литералы | Интерпретируются как string, заключаются в двойные кавычки (“Hello ”), являются rvalue. |
| Литералы логического типа | Интерпретируются как logic, принимают значения true и false. |

## **1.9. Объявление данных**

Область видимости «сверху вниз» (по принципу С++). В языке MTD-2020 требуется обязательное объявление переменной перед её использованием, объявление данных начинается с указания типа данных, затем указывается имя идентификатора.

Примеры: uwhole a, string b, logic c;

## **1.10. Инициализация данных**

При объявлении переменной допускается инициализация данных. При этом переменной будет присвоено значение литерала или идентификатора, стоящего справа от знака равенства (:). Объектами-инициализаторами могут быть идентификаторы, литералы, выражения и вызовы функций. Для инициализации целочисленных литералов, представленных в восьмеричной системе счисления, предусмотрен префикс 00, который ставится перед литералом.

## **1.11. Инструкции языка**

В языке MTD-2020 предусмотрены следующие инструкции. Представлены в таблице. Инструкции языка MTD-2020 представлена в таблице в таблице 1.4.

Таблица 1.4 – Инструкции языка

|  |  |
| --- | --- |
| Инструкция | Форма записи |
| Объявление переменной | <тип данных> <идентификатор>. |
| Главная функция | master  [  /программный блок/  ] |
| Объявление функции | <тип данных> func <идентификатор> (<тип данных> <идентификатор>, …)  [  / программный блок /  repay <идентификатор/литерал>.  ] |
| Вызов функции | <идентификатор> (<идентификатор>, ...) |
| Присвоение значения | : |

Продолжение таблицы 1.4

|  |  |
| --- | --- |
| Печать данных | meout <литерал/идентификатор>. |
| Функции стандартной библиотеки  (применяются при инициализации) | strdup(string) — копирует содержиое из строки string.  stradd(string1, string2) — производит конкатенацию двух строк. |
| Возвращаемое значение | repay <литерал/идентификатор>. |
| Инструкция условного оператора | if (условие)  [  <блок операций1>  ]  else  [  <блок операций2>  ] |

## **1.12. Операции языка**

Арифметические операции, которые можно использовать в языке MTD-2020, представлены в таблице 1.5.

Таблица 1.5 — Операции языка

|  |  |
| --- | --- |
| Инструкция | Форма записи |
| Арифметические операции языка | + — бинарное сложение  − — бинарный минус  \* — бинарное произведение  / — бинарное деление  ( ) — приоритет операций |
| Побитовые операции языка | | — побитовое или  & — побитовое и  ~ — побитовая инверсия |

Приоритет подчиняется правилам:

* если в вы­ра­же­ние без ско­бок вхо­дят толь­ко сло­же­ние и вы­чи­та­ние или толь­ко умно­же­ние и де­ле­ние, то дей­ствия вы­пол­ня­ют в том по­ряд­ке, в каком они на­пи­са­ны;
* если в вы­ра­же­ние без ско­бок вхо­дят не толь­ко дей­ствия сло­же­ния и вы­чи­та­ния, но и умно­же­ния и де­ле­ния, или оба этих дей­ствия, то сна­ча­ла вы­пол­ня­ют по по­ряд­ку (слева на­пра­во) умно­же­ние и де­ле­ние, а затем сло­же­ние и вы­чи­та­ние;
* если в вы­ра­же­нии име­ют­ся скоб­ки, то сна­ча­ла вы­чис­ля­ют зна­че­ние вы­ра­же­ний в скоб­ках.

## **1.13. Выражения и их вычисление**

Правила для выражения и их вычисления в языке MTD-2020:

* выражение записывается только в одну строку;
* операторы разделяются пробелами;
* допускается использование скобок (для изменения приоритета);
* не допускается запись двух подряд идущих арифметических операций;

## **1.14. Конструкции языка**

Основные программные конструкции языка MTD-2020 представлены в таблице 1.6.

Таблица 1.6 — Основные конструкции языка

|  |  |
| --- | --- |
| Конструкция | Реализация |
| Главная функция  (точка входа) | master  [  / программный блок /  ] |
| Функции | <тип данных> func <идентификатор>  (<тип данных> <идентификатор>, …)  [  / программный блок /  repay <идентификатор/литерал>.  ]  <тип данных> func <идентификатор> ()  [  repay <идентификатор/литерал>.  ] |

## **1.15. Область видимости идентификаторов**

Область видимости идентификаторов в языке MTD-2020 – «сверху вниз» (по принципу С++). Это значит, что к каждому идентификатору можно обратиться только ниже описания его описания или объявления. «Ниже» понимается буквально — ниже по тексту программы. Т.е., например, до описания функции ее нельзя вызывать.

Все переменные в языке MTD-2020 обязаны находится внутри программного блока функций. Объявление глобальных переменных (вне программных блоков функций) не предусмотрено.

## **1.16. Семантические проверки**

Основные семантические правила языка MTD-2020 проверяемые на этапах работы транслятора, представлены в таблице 1.7.

Таблица 1.7 — Семантические правила

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Должна присутствовать точка входа master и только одна |
| 3 | Не должно быть объявлений идентификаторов с одинаковыми именами в одном и том же блоке кода |
| 4 | Присваивать значение идентификатору можно только соответствующего типа |
| 5 | Вызов функции обязует использование скобок после ее названия с передачей параметров соответствующих типов или без них |
| 6 | Тип возвращаемого функцией значения должен соответствовать типу функции |
| 7 | Деление на ноль запрещено |
| 8 | Присваивать значение функции запрещено |
| 9 | Проводить арифметические операции со строковым типом данных запрещено |
| 10 | Неверно введено восьмеричное представление числа |

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

На этапе выполнения все переменные данного языка помещаются в стек.

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

Язык MTD-2020 содержит стандартную билбиотек mtd. Подключение стандартной библиотеки обязательно перед использованием функции. Пример функция для данной билбиотеки представлена в таблице 1.8.

Таблица 1.8 — Библиотека

|  |  |
| --- | --- |
| Функция | Описание |
| stradd (string1, string2). | конкатенация двух строк; |
| strdup(string). | копирование строки string |

## **1.19. Ввод и вывод данных**

Ввод данных языком программирования MTD-2020 не поддерживается.

Для вывода данных используется функция meout(<имя идентификатора>).

Пример: meout (a);

## **1.20. Точка входа**

В языке MTD-2020 может быть только одна точка входа и определяется наличием функции master**.** При инициализации более одной или менее одной – выдаст ошибку лексического анализатора.

## **1.21. Препроцессор**

Препроцессор, принимающий и выдающий некоторые данные на вход транслятору, в языке MTD-2020 отсутствует.

## **1.22. Соглашения о вызовах**

Соглашение о вызовах — это протокол для передачи аргументов функциям. Другими словами, это договоренность между вызывающим и вызываемым кодом.

В языке MTD-2020 по умолчанию применяется соглашение \_stdcall, где параметры помещаются в стек, передача параметров происходит справа налево, стек освобождает вызываемый код, возврат через регистр EAX.

## **1.23. Объектный код**

Целевой язык для языка программирования MTD-2020 является язык JavaScript, после чего интерпретируется браузером.

## **1.24. Классификация сообщений транслятора**

В ходе своей работы, транслятор генерирует сообщения, которые информируют пользователя о допущенных ошибках. Все сообщения транслятора разделены на интервалы, в зависимости от того на каком этапе была обнаружена ошибка. Все интервалы представлены в таблице 1.9.

Таблица 1.9– Сообщения транслятора

|  |  |
| --- | --- |
| Интервал | Описание |
| 100-104 | Ошибки входных параметров |
| 106-112 | Ошибки открытия и чтения файлов |
| 113-122 | Ошибки на этапе лексического анализа |
| 600-609 | Ошибки на этапе синтаксического анализа |
| 700-711 | Ошибки на этапе семантического анализа |
| 800 | Ошибка на этапе генерации в язык JavaScript |

## **1.25. Контрольный пример**

Контрольный пример языка программирования MTD-2020 представлен в приложении А.

# **Глава 2 Структура транслятора**

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

Транслятор языка MTD-2020 преобразует исходный код программы в код целевого языка. Процесс трансляции состоит из фаз: лексический анализ, синтаксический анализ, семантический анализ и генерация кода. Алгоритм выполнения и описание каждой фазы представлено в таблице 2*.*1. На всех фазах трансляции применяется таблица идентификаторов и таблица лексем. Графически схема транслятора представлена на рисунке 2.1.



Рисунок 2.1 — Схема структуры транслятора

Таблица 2.1 — Структура транслятора

|  |  |
| --- | --- |
| Наименование подпрограммы | Предназначение |
| Лексический анализатор | Обрабатывает входной файл исходного кода, проверяя его на разрешённые, запрещённые и игнорируемые символы. Преобразует исходный код в более простой с помощью замены длинных слов (идентификаторов, типов данных, ключевых слов и т.д.) на лексемы, состоящие из одного символа, что упрощает последующую работу с кодом. После обработки входного файла формируется таблица лексем и таблица идентификаторов. |

Продолжение таблицы 2.1

|  |  |
| --- | --- |
| Синтаксический анализатор | Проверяет правильность написанных конструкций по заданной грамматике, выявляет синтаксические ошибки при их наличии и формирует дерево разбора. |
| Семантический анализатор | Проверяет правильность исходного кода с точки зрения семантики. Реализован как отдельный этап в данном трансляторе. Проверка по правилам осуществляется после лексического и синтаксического анализа. |
| Генератор кода | Этап транслятора, выполняющий генерацию кода на языке MTD-2020 в код на языке JavaScript на основе данных, полученных на предыдущих этапах трансляции. |

## **Перечень входных параметров транслятора**

Благодаря входным параметрам транслятора осуществляется контроль за выводом протоколов работы разных компонент, а также указывается файл с исходным кодом — программы. Перечень входных параметров представлена в таблице 2.2.

Таблица 2.2 — Входные параметры транслятора

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Тип параметра |
| -in: | Указывает транслятору путь к исходному коду. | Обязательный. |
| -log: | Указывает транслятору в какой файл выводить протокол работы транслятора. | Не обязательный. |
| -lx: | Указывает транслятору в какой файл выводить результат работы лексического анализатора, а именно таблицу лексем и таблицу идентификаторов. | Не обязательный. |
| -id: | Указывает транслятору в какой файл выводить таблицу идентификаторов | Не обязательный. |
| -js: | Указывает транслятору в какой файл выводить результат работы компилятора | Не обязательный. |
| -rl: | Указывает транслятору в какой | Не обязательный. |

Продолжение таблицы 2.2

|  |  |  |
| --- | --- | --- |
|  | файл выводить результат работы синтаксического анализатора, а именно трассировку и дерево разбора. |  |

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

Транслятор языка MTD-2020 формирует один протокол его работы. Информация, записываемая в протокол, представлена в таблице 2.3.

Таблица 2.3 — Протокол транслятора.

|  |  |
| --- | --- |
| Тип информации | Описание информации |
| Дата и время трансляции | Выводится дата и время. |
| Параметры командой строки | Выводится информация об указанных параметрах командной строки. |
| Полная таблица лексем | Выводится таблица лексем с информацией к каждой лексеме. |
| Таблица идентификаторов | Выводится таблица идентификаторов с дополнительной информацией. |
| Промежуточный код | Выводится исходный код, представленный в виде лексем. |
| Трассировочная информация синтаксического анализа | Выводится полная информация о разборе таблицы лексем на синтаксическом анализаторе. |
| Дерево разбора | Выводится правила, по которым осуществился разбор исходного кода. |
| Промежуточный код с использованием польской записи | Выводится исходный код, представленный в виде лексем, выражения представлены в польской записи. |

# **Глава 3 Разработка лексического анализатора**

## **Структура лексического анализатора**

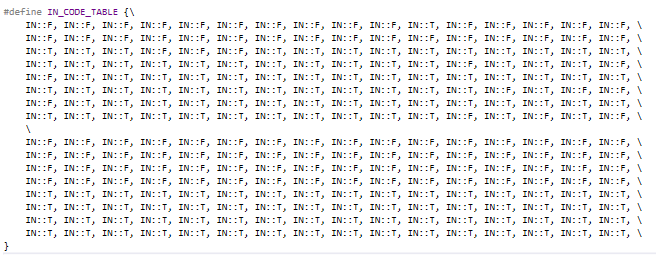
Лексический анализатор — часть транслятора, выполняющая лексический анализ, т.е. преобразующая исходный текст, заменяя лексические единицы языка их внутренним представлением — лексемами. Входными данными для лексического анализатора является предварительно обработанный текст программы на языке MTD-2020. В результате работы лексического анализатора формируется таблица идентификаторов и таблица лексем, модифицируется протокол работы транслятора. Структура лексического анализатора MTD-2020 представлена на рисунке 3.1.



Рисунок 3.1 — Структура лексического анализатора MTD-2020

* 1. **Контроль входных символов**

Таблица для контроля входных символов представлена на рисунке 3.2.

Рисунок 2.2 — Структура лексического анализатора MTD-2020

Принцип работы таблицы: каждому элементу соответствует значение в шестнадцатеричной системе счисления - такое же, как и в таблице Windows-1251 (см. рисунок 1.1).

В представленной таблице: F — запрещённый символ, T — разрешённый символ, I — игнорируемый символ.

## **Удаление избыточных символов**

Удаление избыточных символов в языке в языке MTD-2020 не поддерживается.

## **Перечень ключевых слов, сепараторов, символов операций соответствующим им лексемам и конечных автоматов**

Перечень ключевых слов, сепараторов, символов операций соответствующих им лексем представлен в таблице 3.1.

Таблица 3.1 - Перечень ключевых слов

|  |  |
| --- | --- |
| Цепочка | Лексема |
| uwhole | t |
| string | t |
| logic | f |
| meout | p |
| repay | r |
| master | m |
| if | w |
| else | e |
| func | f |
| . | . |
| [ | [ |
| ] | ] |
| ( | ( |
| ) | ) |

Продолжение таблицы 3.1

|  |  |
| --- | --- |
| > | > |
| < | < |
| = | = |
| : | : |
| + | + |
| - | - |
| \* | \* |
| / | / |
| & | & |
| | | | |
| ~ | ~ |
| ; | ; |
| идентификатор | i |
| числовой литерал | l |
| строковый литерал | l |
| логический литерал | l |
| stradd, strdup | s |

* 1. **Основные структуры данных**

Основными структурами данных лексического анализатора языка MTD-2020 являются таблица лексем и таблица идентификаторов. В таблице лексем содержится лексема, ее номер, полученный при разборе, номер строки и позиции в исходном коде, индекс в таблице идентификаторов, а также приоритет операции и её знак, если лексема является операцией. В таблице идентификаторов содержится имя идентификатора, номер в таблице лексем, тип данных, тип идентификатора, а также его значение.

## **Принцип обработки ошибок**

При обнаружении ошибки во время работы транслятора, вызывается функция получения ошибки, в которую передается, в зависимости от места возникновения ошибки, следующая информация: код ошибки, номер строки в коде и сообщение о типе ошибки. При возникновении ошибки работа транслятора прекращается.

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

Перечень сообщений, формируемых лексическим анализатором в ходе своей работы, представлен в таблице 3.2.

Таблица 3.2 — Перечень сообщений лексического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 113 | Превышен максимальный размер таблицы лексем |
| 114 | Превышен максимальный размер таблицы идентификаторов |
| 115 | Недопустимы номер возвращаемой строки таблицы лексем |
| 117 | Отсутствует таблица лексем |
| 118 | Отсутствует таблица идентификаторов |
| 119 | Недопустимы номер возвращаемой строки таблицы идентификаторов |
| 120 | Размер длины строки превышен(256) |
| 121 | Превышено максимальное значение целочисленного литерала |
| 122 | Превышен максимальный размер имени идентификатора |

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

Транслятор допускает использование параметра для управления работой лексического анализатора, а именно выводом таблицы лексем и таблицы идентификаторов. Описание параметров представлено в таблице 2.2.

## **Алгоритм лексического анализа**

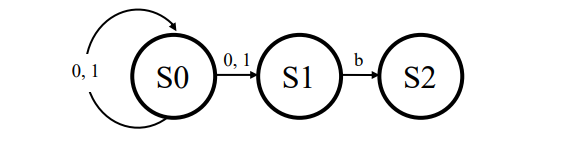
Лексический анализ является первой и наиболее простой фазой трансляции. Алгоритм лексического анализатора заключается в следующем: после разбиения текста из файла с исходным кодом на слова, для каждого слова подбирается конечный автомат, способный его разобрать, в случае если такой автомат существует, цепочка будет разобрана, иначе ошибка. Далее лексический анализатор анализирует лексему, соответствующую данному слову, и выполняет действия, описанные для данной лексемы. Лексический анализатор продолжает работать пока не будет разобрано последнее слово. 

Рисунок 3.5 – Граф переходов для двоичного целочисленного литерала.

Работу конечных автоматов можно представить в виде графа.

## **Контрольный пример**

Результатом работы лексического анализатора являются таблица лексем и таблица идентификаторов. Содержимое таблиц на основе исходного кода из приложения А представлено в приложении В.

# **Глава 4 Разработка синтаксического анализатора**

## **Структура синтаксического анализатора**

Синтаксический анализатор является второй фазой работы транслятора. Назначением синтаксического анализаторы является распознавание синтаксических конструкций языка и формирование промежуточного кода. Исходными данными синтаксического анализатора являются таблицы лексем и идентификаторов. Свою работу начинает только при условии отсутствии ошибок в их общей структуре. Лексемы являются для синтаксического анализатора терминальными символами контекстно-свободной грамматики. Если в ходе его работы не было обнаружено ошибок, то формируется дерево разбора (промежуточное представление кода) и модифицируется протокол работы. Структура синтаксического анализатора представлена на рисунке 4.1.



Рисунок 4.1 — Структура синтаксического анализатора MTD-2020

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

Каждый язык программирования описывается с помощью набора правил, определяющих структуру правильной программы. Наиболее удобным формализмом для описания синтаксических конструкций языка программирования являются контекстно-свободные грамматики.

Грамматика для синтаксического разбора языка MTD-2020 представляется четверкой G = <T, N, P, S>, где Т — множество терминальных символов, N — множество нетерминальных символов, P — множество правил языка, S — начальный символ грамматики, являющийся нетерминалом.

В грамматике языка MTD-2020 множество нетерминальных символов представлено следующим образом:

* S — Порождает правила, описывающие общую структуру программы;
* N — Порождает правила, описывающие инструкции языка
* Q — Порождает правила, описывающие формальные параметры функции;
* O — Порождает правила, описывающие параметры в стандартных функциях;
* Y — Порождает правила, описывающие выражения;
* E — Порождает правила, описывающие выражения;
* P — Порождает правила, описывающие арифметические выражения;
* M — Порождает правила, описывающие арифметические операции;
* B — Порождает правила, описывающие операции сравнения;
* K — Порождает правила, описывающие выражения в условном операторе;

Перечень правил, описывавших контекстно-свободную грамматику языка MTD-2020, представлен в таблице 4.1.

Таблица 4.1 – Правила грамматики MTD-2020

|  |  |
| --- | --- |
| Нетерминальный символ | Цепочки правил |
| S→ | tfi(Q)[Y]S | d[N] |
| N→ | ti:E.N | ti:E. | tO.N | tO. | ti:P.N | ti:P. | i:P.N | i:P. | i:E.N | i:E. | w(K)[N]e[N]N | w(K)[N]N | w(K)[N]e[N] | w(K)[N] | p(i).N | p(i). | p(l).N | p(l). |
| Q→ | ti | ti,Q | |
| O→ | i | i,O |
| Y→ | ti:E.Y | ti:E. | tO.Y | tO. | ti:P.Y | ti:P. | i:P.Y | i:P. | i:E.Y | i:E. | w(K)[Y]e[Y]Y | w(K)[Y]Y | w(K)[Y]e[Y] | w(K)[Y] | p(i).Y | p(i). | p(l).Y | p(l). | ri. | rli |
| E→ | i | l | s(i) | s(i,i) | s(l) | s(l,l) | s(O) | ~i | ~l |
| P→ | i | l | iP | lP | iMP | lMP | (P) | (P)MP | (P)N | (P) |
| M→ | + | - | \* | / | & | | | ~ |
| B→ | > | < | = |
| K→ | l | i | iBi | iBl | lBi | lBl |

## **Построение конечного магазинного автомата**

Конечный автомат с магазинной памятью представляет собой семерку,где Q — множество состояний автомата, V — алфавит входных символов, Z – алфавит специальных магазинных символов,  — функция переходов автомата,  — начальное состояние автомата,  — начальное состояние магазинного автомата, F — множество конечных состояний.

Пример работы магазинного автомата для цепочки представлен на рисунке 4.2.

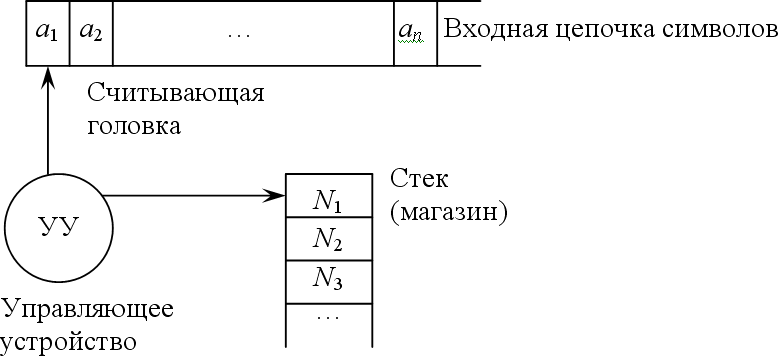


Рисунок 4.2 — Автомат с магазинной памятью

## **Основные структуры данных**

Основные структуры данных синтаксического анализатора включают в себя структуру магазинного автомата и структуру грамматики Грейбах, описывающая правила языка MTD-2020. Структура синтаксического анализатора представлена в приложении Г.

## **Описание алгоритма синтаксического разбора**

Входные символы и лексемы в форме Грейбах находятся в ленте на входе конечного автомата.

1) Если лента не пустая, переходим далее следующему пункту, иначе переходим к пункту 5.

2) Если на верхушке магазина нетерминальный символ.

2.1) Если есть такое правило, то переходим к следующему пункту.

2.1.1) Если цепочка есть, возвращаем NS\_OK. Переходим к пункту 4.

2.1.2) Иначе восстанавливаем состояние. Переходим к пункту 4.

2.2) Иначе возвращаем ошибку. Переход к пункту 4.

3) Если на верхушке терминал и он совпадает с символом на ленте, то удаляем его из стека и продвигаем ленту. Переход к пункту 4.

4) Повторяем шаг, переходим к пункту 1.

5) Конец работы.

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

Перечень сообщений синтаксического анализатора представлен в таблице 4.2.

Таблица 4.2 — Перечень сообщений синтаксического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 600 | Неверная структура программы |
| 601 | Ошибка в написании инструкций или конструкций языка |
| 602 | Ошибка инициализации |
| 603 | Ошибка выражения |
| 604 | Ошибка знака в выражении |
| 605 | Ошибка знака в сравнении |
| 606 | Ошибка в условии if |
| 607 | Ошибка объявления |
| 608 | Ошибка параметров функции |
| 609 | Ошибки в теле функции |

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

Транслятор допускает использование параметра для управления работой синтаксического анализатора, а именно вывода дерева разбора и трассировки. Описание параметров представлено в таблице 2.2.

## **Принцип обработки ошибок**

Принцип заключается в том, что синтаксический анализатор перебирает все возможные правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем. В случае если не была найдена ни одна подходящая цепочка, то формируется соответствующая ошибка из таблицы 4.2. Все ошибки записываются в общую структуру ошибок, а также отображаются на консоли.

## **Контрольный пример**

Из контрольного примера, представленного в приложении А, результатом работы синтаксического анализатора, является трассировка и дерево разбора, представленные в приложении Д.

# **Глава 5 Разработка семантического анализатора**

## **Структура семантического анализатора**

Семантический анализатор в трансляторе языка MTD-2020 выделен в отдельную фазу. Проверка на ошибки в исходном коде производится после этапов лексического и синтаксического анализа. Общая структура обособленно работающего (не параллельно с лексическим анализом) семантического анализатора представлена на рисунке 5.1.

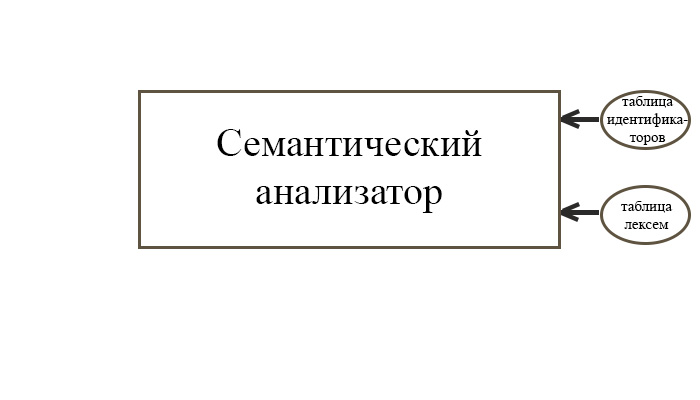


Рисунок 5.1 Структура семантического анализатора

## **5.2** **Функции семантического анализатора**

Семантический анализотор языка MTD-2020 выполняет проверку на основные правила языка (семантики языка), описанные в разделе 1.16.

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

Сообщения, формируемые семантическим анализатором представлены в таблице 5.1.

Таблица 5.1 — Перечень сообщений семантического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 700 | Тип данных передаваемых в функцию должен соответствовать заявленному. |
| 701 | Количество параметров передаваемых в функцию должно соответствовать заявленному значению. |
| 702 | Тип данных справа в выражении должен совпадать с типом слева |
| 703 | Для сравнения в блоке if без знаков сравнения могут использоваться только литералы и идентификаторы типа logic |
| 704 | Деление в выражении на ноль. |
| 705 | В программе может быть только одна точка входа |
| 706 | Применение знаков арифметических операций к идентификаторам или литералам типа string или logic |
| 707 | Применение знаков логических операций к идентификаторам или литералам типа string или logic |

Продолжение таблицы 5.1

|  |  |
| --- | --- |
| 708 | Тип данных возвращаемого значения не соответствует типу данных функции |
| 709 | Повторное объявление переменной |
| 710 | Использование необъявленной переменной |
| 711 | Неверно введено восьмеричное представление числа |

## **5.4 Принцип обработки ошибок**

Принцип обработки ошибок идентичен принципу обработки ошибок на этапе лексического анализа (раздел 3.6).

# 

# **Глава 6 Вычисление выражений**

## **Выражения, допускаемые языком**

Выражения, допускаемые языком MTD-2020, выполняются над целочисленными типами данных. В выражениях поддерживаются арифметические операции, такие как +, -, \*, / , &, | и (), и вызовы функций из них.

Приоритетность арифметических операций представлена в таблице 6.1.

Таблица 6.1 — Приоритетность операций

|  |  |  |
| --- | --- | --- |
| Операция | Приоритет | Порядок выполнения |
| + | 2 | Наименьший приоритет |
| - | 2 | Наименьший приоритет |
| | | 2 | Наименьший приоритет |
| & | 3 | Вторая степень приоритетности |
| \* | 3 | Вторая степень приоритетности |
| / | 3 | Вторая степень приоритетности |
| ( | 0 | Наивысший приоритет |
| ) | 0 | Наивысший приоритет |

## **Польская запись и принцип ее построения**

Польская запись — это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок. Существует два типа польской записи: прямая и обратная, также известные как префиксная и постфиксная.

Обратная польская запись — это форма записи математических выражений, в которой операторы расположены после своих операндов. Выражение в обратной польской нотации читается слева направо: операция выполняется над двумя операндами, непосредственно стоящими перед знаком этой операции. Результат операции заменяет в выражении последовательность её операндов и символ операции. Результатом вычисления всего выражения является результат последней вычисленной операции.

Алгоритм построения польской записи:

* исходная строка: выражение;
* результирующая строка: польская запись;
* стек: пустой;
* исходная строка просматривается слева направо;
* операнды переносятся в результирующую строку;
* операция записывается в стек, если стек пуст;
* операция выталкивает все операции с большим или равным приоритетом в результирующую строку;
* отрывающая скобка помещается в стек;
* закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются.
* по концу разбора исходной строки все операции, оставшиеся в стеке, выталкиваются в результирующую строку.

## **Программная реализация обработки выражений**

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

## **Контрольный пример**

Результаты преобразования выражений исходного кода в польскую запись представлены в приложении Е. Ход разбора одного из выражений в контрольном примере в польский формат записи приведен в таблице 6.2.

Таблица 6.2 — Пример конвертации выражения в польскую запись

|  |  |  |
| --- | --- | --- |
| Выражение | Стек | Результат |
| ((45 + 25) / 2) \* (y - 2) & 4 |  |  |
| (45 + 25) / 2) \* (y - 2) & 4 | ( |  |
| 45 + 25) / 2) \* (y - 2) & 4 | (( |  |
| + 25) / 2) \* (y - 2) & 4 | (( | 45 |
| 25) / 2) \* (y - 2) & 4 | ((+ | 45 |
| ) / 2) \* (y - 2) & 4 | ((+ | 45 25 |
| / 2) \* (y - 2) & 4 | ( | 45 25 + |
| 2) \* (y - 2) & 4 | (/ | 45 25 + |
| ) \* (y - 2) & 4 | (/ | 45 25 + 2 |
| \* (y - 2) & 4 |  | 45 25 + 2 / |
| (y - 2) & 4 | \* | 45 25 + 2 / |
| y - 2) & 4 | \*( | 45 25 + 2 / |
| - 2) & 4 | \*( | 45 25 + 2 / y |

Продолжение таблицы 6.2

|  |  |  |
| --- | --- | --- |
| 2) & 4 | \*(- | 45 25 + 2 / y |
| ) & 4 | \*(- | 45 25 + 2 / y 2 |
| & 4 |  | 45 25 + 2 / y 2 - \* |
| 4 | & | 45 25 + 2 / y 2 - \* |
|  | & | 45 25 + 2 / y 2 - \* 4 |
|  |  | 45 25 + 2 / y 2 - \* 4 & |

# 

# **Глава 7 Генерация кода**

## **7.1 Структура генератора кода**

Генерация объектного кода — это перевод транслятором представления исходной программы на языке MTD-2020 в цепочку символов выходного языка JavaScript. На вход генератора подаются таблицы лексем и идентификаторов, а также исходный код, разбитый на слова, на основе которых генерируется файл с кодом на языке JavaScript, который впоследствии будет интерпретироваться браузером. Схематично генерация кода показана на рисунке 7.1.Генератор кода начинает свою работу только в том случае, если код на языке MTD-2020 прошёл предыдущие этапы без ошибок.

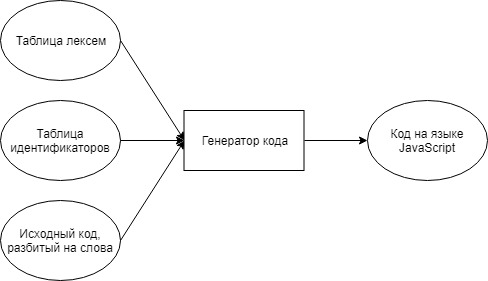


Рисунок 7.1 — Структура генератора кода

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

Язык MTD-2020 требует указывать тип данных при объявлении идентификатора. Язык JavaScript не обязывает указывать тип, объявление происходит с указанием ключевого слова var и имени переменной. Соответствия между типами данных идентификаторов языка MTD-2020 и языка JavaScript представлены в таблице 7.1.

Таблица 7.1 — Соответствие типов данных

|  |  |
| --- | --- |
| Тип данных на языке MTD-2020 | Тип данных на языке JavaScript |
| uwhole(целочисленный) | Number (числовой) |
| string (строковый) | String (строковый) |
| logic (логический) | Boolean (логический) |

## **7.3 Статическая библиотека**

Для языка MTD-2020 статическая библиотека как отдельный файл не реализована. Преобразование функций происходит на этапе генерации кода.

## **7.4 Особенности алгоритма генерации кода**

Алгоритм генерации исходного кода на языке MTD-2020:

* открытие файла “Code.js” для записи;
* открытие файла “IntoJSGeneration.html”
* на вход генератора подаются таблицы лексем и идентификаторов, а также исходный код, разбитый на слова;
* каждый элемент таблицы лексем проверяется на соответствие с зарезервированными лексемами;
* если соответствие найдено, в выходной файл “Code.js” записывается соответствующее выражение. Пример данной операции для двух лексем представлено далее;
* когда каждая лексема из таблицы разобрана, код сгенерирован, файл закрывается;

Пример операции для двух лексем:

if (ltable.table[i].lexema == LEX\_RIGHTHESIS)

{

\*js.stream << ")";

}

if (ltable.table[i].lexema == LEX\_LEFTBRACE)

{

\*js.stream << "{";

}

}

Часть кода генерации

## **7.5 Входные параметры генератора кода**

На этапе генерации кода транслятор не допускает использование параметров. По умолчанию файлом для сгенерированного кода является файл “Code.js”. Проверку правильности генерации кода можно осуществить в любом html-файле, в который необходимо подключить сгенерированный код:

<script src="Code.js"></script>.

## **7.6 Контрольный пример**

Результат генерации кода на основе контрольного примера представлен в приложении Ж.

# **Глава 8 Тестирование транслятора**

## **8.1 Общие положения**

В результате обработки исходного кода программы, представленного в приложении А, транслятор языка MTD-2020 генерирует общий протокол работы, куда записываются все возникшие ошибки и предупреждения. Кроме того, все ошибки, возникшие на этапах лексического и семантического анализов, выводятся на консоль. Из ошибок, возникших на этапе синтаксического анализа, на консоль выводится только первая.

## **8.2 Результаты тестирования**

Транслятор языка MTD-2020 представляет диагностику и выявление ошибок на разных этапах трансляции. Ниже будут приведены результаты обработки транслятором исходного кода с допущенными ошибками.

Тестирование ошибок транслятора представлено в таблице 8.1. В таблице 8.1 приведены три вида ошибок, первая происходит на этапе лексического анализа, вторая — синтаксического, третья — семантического.

Таблица 8.1 — Тестирование

|  |  |
| --- | --- |
| Исходный код с ошибкой | Генерируемое сообщение об ошибке |
| master  [  uwhole a : 321.  meout(a).  ] | Ошибка 121: Превышено максимальное значение целочисленного литерала  Строка: 2, символ: 13 |
| master  uwhole y : 11.  meout (y).  ] | Ошибка 600: Неверная структура программы  Строка: 4, символ: 8 |
| master  [  uwhole c : "string".  meout (c).  ] | Ошибка 702: Тип данных справа в выражении должен совпадать с типом слева  Строка: 3, символ: 16 |

# **Заключение**

В ходе выполнения курсовой работы был разработан транслятор для языка программирования MTD-2020. Таким образом, были выполнены основные задачи данной курсовой работы:

* сформулирована спецификация языка MTD-2020;
* разработаны конечные автоматы и алгоритмы для реализация лексического анализатора;
* разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
* разработан семантический анализатор, осуществляющий проверку смысла используемых инструкций;
* разработан транслятор с языка программирования MTD-2020 на язык JavaScript;
* проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка MTD-2020 включает:

* 3 типа данных;
* Поддержка операции вывода;
* Возможность вызова функций стандартной библиотеки;
* Наличие 4 арифметических операторов и 3 побитовых для вычисления выражений;
* Наличие условного оператора;
* Структурированная система для обработки ошибок пользователя.

# **Список использованных источников**

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

2. Разработка компиляторов / А.А. Терехов, А.Е. Москаль, Д.Ю. Булычев, Н.Н. Вояковская, 2016. – 375с.

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

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

# **Приложение А**

uwhole func myuf(uwhole a)

[

uwhole t : 0033.

t : t + a.

repay t.

]

master

[

logic check : true.

meout(check).

string str : "hello".

string sss : strdup(str).

meout(sss).

uwhole m : myuf(3).

meout(m).

uwhole inverse : ~ 15.

meout(inverse).

uwhole or : 55 | 80.

uwhole and : 56 & 76.

if (or > and)

[

meout(or).

]

else

[

meout(and).

]

string n : "hello".

string s : "world".

string ns : stradd(n, s).

meout(ns).

uwhole j : ((44 + 4) / 7).

meout(j).

]

# **Приложение Б**

#define FST\_UWHOLE 7, \

FST::NODE(1, FST::RELATION('u', 1)),\

FST::NODE(1, FST::RELATION('w', 2)),\

FST::NODE(1, FST::RELATION('h', 3)),\

FST::NODE(1, FST::RELATION('o', 4)),\

FST::NODE(1, FST::RELATION('l', 5)),\

FST::NODE(1, FST::RELATION('e', 6)),\

FST::NODE()

#define FST\_STRING 7, \

FST::NODE(1, FST::RELATION('s', 1)),\

FST::NODE(1, FST::RELATION('t', 2)),\

FST::NODE(1, FST::RELATION('r', 3)),\

FST::NODE(1, FST::RELATION('i', 4)),\

FST::NODE(1, FST::RELATION('n', 5)),\

FST::NODE(1, FST::RELATION('g', 6)),\

FST::NODE()

#define FST\_LOGIC 6, \

FST::NODE(1, FST::RELATION('l', 1)),\

FST::NODE(1, FST::RELATION('o', 2)),\

FST::NODE(1, FST::RELATION('g', 3)),\

FST::NODE(1, FST::RELATION('i', 4)),\

FST::NODE(1, FST::RELATION('c', 5)),\

FST::NODE()

#define FST\_FUNC 5, \

FST::NODE(1, FST::RELATION('f', 1)),\

FST::NODE(1, FST::RELATION('u', 2)),\

FST::NODE(1, FST::RELATION('n', 3)),\

FST::NODE(1, FST::RELATION('c', 4)),\

FST::NODE()

#define FST\_REPAY 6, \

FST::NODE(1, FST::RELATION('r', 1)),\

FST::NODE(1, FST::RELATION('e', 2)),\

FST::NODE(1, FST::RELATION('p', 3)),\

FST::NODE(1, FST::RELATION('a', 4)),\

FST::NODE(1, FST::RELATION('y', 5)),\

FST::NODE()

#define FST\_MEOUT 6, \

FST::NODE(1, FST::RELATION('m', 1)),\

FST::NODE(1, FST::RELATION('e', 2)),\

FST::NODE(1, FST::RELATION('o', 3)),\

FST::NODE(1, FST::RELATION('u', 4)),\

FST::NODE(1, FST::RELATION('t', 5)),\

FST::NODE()

#define FST\_MASTER 7, \

FST::NODE(1, FST::RELATION('m', 1)),\

FST::NODE(1, FST::RELATION('a', 2)),\

FST::NODE(1, FST::RELATION('s', 3)),\

FST::NODE(1, FST::RELATION('t', 4)),\

FST::NODE(1, FST::RELATION('e', 5)),\

FST::NODE(1, FST::RELATION('r', 6)),\

FST::NODE()

#define FST\_IF 3, \

FST::NODE(1, FST::RELATION('i',1)),\

FST::NODE(1, FST::RELATION('f',2)),\

FST::NODE()

#define FST\_ELSE 5, \

FST::NODE(1, FST::RELATION('e', 1)),\

FST::NODE(1, FST::RELATION('l', 2)),\

FST::NODE(1, FST::RELATION('s', 3)),\

FST::NODE(1, FST::RELATION('e', 4)),\

FST::NODE()

#define FST\_STRDUP 7,\

FST::NODE(1, FST::RELATION('s',1)),\

FST::NODE(1, FST::RELATION('t',2)),\

FST::NODE(1, FST::RELATION('r',3)),\

FST::NODE(1, FST::RELATION('d',4)),\

FST::NODE(1, FST::RELATION('u',5)),\

FST::NODE(1, FST::RELATION('p',6)),\

FST::NODE()

#define FST\_STRADD 7,\

FST::NODE(1, FST::RELATION('s',1)),\

FST::NODE(1, FST::RELATION('t',2)),\

FST::NODE(1, FST::RELATION('r',3)),\

FST::NODE(1, FST::RELATION('a',4)),\

FST::NODE(1, FST::RELATION('d',5)),\

FST::NODE(1, FST::RELATION('d',6)),\

FST::NODE()

#define FST\_ID 2, \

FST::NODE(52, \

FST::RELATION('a', 0), FST::RELATION('b', 0), FST::RELATION('c', 0), FST::RELATION('d', 0), FST::RELATION('e', 0), FST::RELATION('f', 0),\

FST::RELATION('g', 0), FST::RELATION('h', 0), FST::RELATION('i', 0), FST::RELATION('j', 0), FST::RELATION('k', 0), FST::RELATION('l', 0),\

FST::RELATION('m', 0), FST::RELATION('n', 0), FST::RELATION('o', 0), FST::RELATION('p', 0), FST::RELATION('q', 0), FST::RELATION('r', 0),\

FST::RELATION('s', 0), FST::RELATION('t', 0), FST::RELATION('u', 0), FST::RELATION('v', 0), FST::RELATION('w', 0), FST::RELATION('x', 0),\

FST::RELATION('y', 0), FST::RELATION('z', 0),\

\

FST::RELATION('a', 1), FST::RELATION('b', 1), FST::RELATION('c', 1), FST::RELATION('d', 1), FST::RELATION('e', 1), FST::RELATION('f', 1),\

FST::RELATION('g', 1), FST::RELATION('h', 1), FST::RELATION('i', 1), FST::RELATION('j', 1), FST::RELATION('k', 1), FST::RELATION('l', 1),\

FST::RELATION('m', 1), FST::RELATION('n', 1), FST::RELATION('o', 1), FST::RELATION('p', 1), FST::RELATION('q', 1), FST::RELATION('r', 1),\

FST::RELATION('s', 1), FST::RELATION('t', 1), FST::RELATION('u', 1), FST::RELATION('v', 1), FST::RELATION('w', 1), FST::RELATION('x', 1),\

FST::RELATION('y', 1), FST::RELATION('z', 1)),\

FST::NODE()

#define FST\_INTLIT 2, \

FST::NODE(20, \

FST::RELATION('1', 0), FST::RELATION('2', 0), FST::RELATION('3', 0), FST::RELATION('4', 0), FST::RELATION('5', 0), FST::RELATION('6', 0),\

FST::RELATION('7', 0), FST::RELATION('8', 0), FST::RELATION('9', 0), FST::RELATION('0', 0),\

\

FST::RELATION('1', 1), FST::RELATION('2', 1), FST::RELATION('3', 1), FST::RELATION('4', 1), FST::RELATION('5', 1), FST::RELATION('6', 1),\

FST::RELATION('7', 1), FST::RELATION('8', 1), FST::RELATION('9', 1), FST::RELATION('0', 1)),\

FST::NODE()

#define FST\_STRLIT 4, \

FST::NODE(1, FST::RELATION('"', 1)),\

FST::NODE(156, \

FST::RELATION('a', 1), FST::RELATION('b', 1), FST::RELATION('c', 1), FST::RELATION('d', 1), FST::RELATION('e', 1), FST::RELATION('f', 1),\

FST::RELATION('g', 1), FST::RELATION('h', 1), FST::RELATION('i', 1), FST::RELATION('j', 1), FST::RELATION('k', 1), FST::RELATION('l', 1),\

FST::RELATION('m', 1), FST::RELATION('n', 1), FST::RELATION('o', 1), FST::RELATION('p', 1), FST::RELATION('q', 1), FST::RELATION('r', 1),\

FST::RELATION('s', 1), FST::RELATION('t', 1), FST::RELATION('u', 1), FST::RELATION('v', 1), FST::RELATION('w', 1), FST::RELATION('x', 1),\

FST::RELATION('y', 1), FST::RELATION('z', 1), FST::RELATION('1', 1), FST::RELATION('2', 1), FST::RELATION('3', 1), FST::RELATION('4', 1),\

FST::RELATION('5', 1), FST::RELATION('6', 1), FST::RELATION('7', 1), FST::RELATION('8', 1), FST::RELATION('9', 1), FST::RELATION('0', 1),\

\

FST::RELATION('а', 1), FST::RELATION('б', 1), FST::RELATION('в', 1), FST::RELATION('г', 1), FST::RELATION('д', 1), FST::RELATION('е', 1),\

FST::RELATION('ё', 1), FST::RELATION('ж', 1), FST::RELATION('з', 1), FST::RELATION('и', 1), FST::RELATION('й', 1), FST::RELATION('к', 1),\

FST::RELATION('л', 1), FST::RELATION('м', 1), FST::RELATION('н', 1), FST::RELATION('о', 1), FST::RELATION('п', 1), FST::RELATION('р', 1),\

FST::RELATION('с', 1), FST::RELATION('т', 1), FST::RELATION('у', 1), FST::RELATION('ф', 1), FST::RELATION('х', 1), FST::RELATION('ц', 1),\

FST::RELATION('ч', 1), FST::RELATION('ш', 1), FST::RELATION('щ', 1), FST::RELATION('ъ', 1), FST::RELATION('ы', 1), FST::RELATION('ь', 1),\

FST::RELATION('э', 1), FST::RELATION('ю', 1), FST::RELATION('я', 1), FST::RELATION(' ', 1), FST::RELATION('.', 1), FST::RELATION(',', 1),\

FST::RELATION('?', 1), FST::RELATION('!', 1), FST::RELATION(';', 1), FST::RELATION(':', 1), FST::RELATION('-', 1), FST::RELATION(' ', 1), \

\

FST::RELATION('a', 2), FST::RELATION('b', 2), FST::RELATION('c', 2), FST::RELATION('d', 2), FST::RELATION('e', 2), FST::RELATION('f', 2),\

FST::RELATION('g', 2), FST::RELATION('h', 2), FST::RELATION('i', 2), FST::RELATION('j', 2), FST::RELATION('k', 2), FST::RELATION('l', 2),\

FST::RELATION('m', 2), FST::RELATION('n', 2), FST::RELATION('o', 2), FST::RELATION('p', 2), FST::RELATION('q', 2), FST::RELATION('r', 2),\

FST::RELATION('s', 2), FST::RELATION('t', 2), FST::RELATION('u', 2), FST::RELATION('v', 2), FST::RELATION('w', 2), FST::RELATION('x', 2),\

FST::RELATION('y', 2), FST::RELATION('z', 2), FST::RELATION('1', 2), FST::RELATION('2', 2), FST::RELATION('3', 2), FST::RELATION('4', 2),\

FST::RELATION('5', 2), FST::RELATION('6', 2), FST::RELATION('7', 2), FST::RELATION('8', 2), FST::RELATION('9', 2), FST::RELATION('0', 2),\

\

FST::RELATION('а', 2), FST::RELATION('б', 2), FST::RELATION('в', 2), FST::RELATION('г', 2), FST::RELATION('д', 2), FST::RELATION('е', 2),\

FST::RELATION('ё', 2), FST::RELATION('ж', 2), FST::RELATION('з', 2), FST::RELATION('и', 2), FST::RELATION('й', 2), FST::RELATION('к', 2),\

FST::RELATION('л', 2), FST::RELATION('м', 2), FST::RELATION('н', 2), FST::RELATION('о', 2), FST::RELATION('п', 2), FST::RELATION('р', 2),\

FST::RELATION('с', 2), FST::RELATION('т', 2), FST::RELATION('у', 2), FST::RELATION('ф', 2), FST::RELATION('х', 2), FST::RELATION('ц', 2),\

FST::RELATION('ч', 2), FST::RELATION('ш', 2), FST::RELATION('щ', 2), FST::RELATION('ъ', 2), FST::RELATION('ы', 2), FST::RELATION('ь', 2),\

FST::RELATION('э', 2), FST::RELATION('ю', 2), FST::RELATION('я', 2), FST::RELATION(' ', 2), FST::RELATION('.', 2), FST::RELATION(',', 2),\

FST::RELATION('?', 2), FST::RELATION('!', 2), FST::RELATION(';', 2), FST::RELATION(':', 2), FST::RELATION('-', 2), FST::RELATION(' ', 2)),\

\

FST::NODE(1, FST::RELATION('"', 3)),\

FST::NODE()

#define FST\_TRUE 5,\

FST::NODE(1, FST::RELATION('t',1)),\

FST::NODE(1, FST::RELATION('r',2)),\

FST::NODE(1, FST::RELATION('u',3)),\

FST::NODE(1, FST::RELATION('e',4)),\

FST::NODE()

#define FST\_FALSE 6,\

FST::NODE(1, FST::RELATION('f',1)),\

FST::NODE(1, FST::RELATION('a',2)),\

FST::NODE(1, FST::RELATION('l',3)),\

FST::NODE(1, FST::RELATION('s',4)),\

FST::NODE(1, FST::RELATION('e',5)),\

FST::NODE()

#define FST\_POINT 2,\

FST::NODE(1, FST::RELATION('.',1)),\

FST::NODE()

#define FST\_COMMA 2,\

FST::NODE(1, FST::RELATION(',',1)),\

FST::NODE()

#define FST\_LEFTBRACE 2,\

FST::NODE(1, FST::RELATION('[',1)),\

FST::NODE()

#define FST\_RIGHTBRACE 2,\

FST::NODE(1, FST::RELATION(']',1)),\

FST::NODE()

#define FST\_LEFTHESIS 2, \

FST::NODE(1, FST::RELATION('(', 1)),\

FST::NODE()

#define FST\_RIGHTHESIS 2, \

FST::NODE(1, FST::RELATION(')', 1)),\

FST::NODE()

#define FST\_MORE 2, \

FST::NODE(1, FST::RELATION('>', 1)),\

FST::NODE()

#define FST\_LESS 2, \

FST::NODE(1, FST::RELATION('<', 1)),\

FST::NODE()

#define FST\_EQUALLY 2, \

FST::NODE(1, FST::RELATION('=', 1)),\

FST::NODE()

#define FST\_COMPARE 2, \

FST::NODE(1, FST::RELATION(':', 1)),\

FST::NODE()

#define FST\_PLUS 2, \

FST::NODE(1, FST::RELATION('+', 1)),\

FST::NODE()

#define FST\_MINUS 2, \

FST::NODE(1, FST::RELATION('-', 1)),\

FST::NODE()

#define FST\_STAR 2, \

FST::NODE(1, FST::RELATION('\*', 1)),\

FST::NODE()

#define FST\_DIRSLASH 2, \

FST::NODE(1, FST::RELATION('/', 1)),\

FST::NODE()

#define FST\_AND 2, \

FST::NODE(1, FST::RELATION('&', 1)),\

FST::NODE()

#define FST\_OR 2, \

FST::NODE(1, FST::RELATION('|', 1)),\

FST::NODE()

#define FST\_NOT 2, \

FST::NODE(1, FST::RELATION('~', 1)),\

FST::NODE()

**Приложение В**

Таблица лексем:

0. tfi(ti)

1. [

2. ti:l.

3. i:i+i.

4. ri.

5. ]

7. m

8. [

9. ti:l.

10. p(i).

11. ti:l.

12. ti:s(i).

13. p(i).

14. ti:i(l).

15. p(i).

16. ti:~l.

17. p(i).

18. ti:l|l.

19. ti:l&l.

20. w(i>i)

21. [

22. p(i).

23. ]

24. e

25. [

26. p(i).

27. ]

28. ti:l.

29. ti:l.

30. ti:s(i,i).

31. p(i).

32. ti:((l+l)/l).

33. p(i).

34. ]

------Таблица лексем------

№ Лексема № строки

0 t 0

1 f 0

2 i 0

3 ( 0

4 t 0

5 i 0

6 ) 0

7 [ 1

8 t 2

9 i 2

10 : 2

11 l 2

12 . 2

13 i 3

14 : 3

15 i 3

16 + 3

17 i 3

18 . 3

19 r 4

20 i 4

21 . 4

22 ] 5

23 m 7

24 [ 8

25 t 9

26 i 9

27 : 9

28 l 9

29 . 9

30 p 10

31 ( 10

32 i 10

33 ) 10

34 . 10

35 t 11

36 i 11

37 : 11

38 l 11

39 . 11

40 t 12

41 i 12

42 : 12

43 s 12

44 ( 12

45 i 12

46 ) 12

47 . 12

48 p 13

49 ( 13

50 i 13

51 ) 13

52 . 13

53 t 14

54 i 14

55 : 14

56 i 14

57 ( 14

58 l 14

59 ) 14

60 . 14

61 p 15

62 ( 15

63 i 15

64 ) 15

65 . 15

66 t 16

67 i 16

68 : 16

69 ~ 16

70 l 16

71 . 16

72 p 17

73 ( 17

74 i 17

75 ) 17

76 . 17

77 t 18

78 i 18

79 : 18

80 l 18

81 | 18

82 l 18

83 . 18

84 t 19

85 i 19

86 : 19

87 l 19

88 & 19

89 l 19

90 . 19

91 w 20

92 ( 20

93 i 20

94 > 20

95 i 20

96 ) 20

97 [ 21

98 p 22

99 ( 22

100 i 22

101 ) 22

102 . 22

103 ] 23

104 e 24

105 [ 25

106 p 26

107 ( 26

108 i 26

109 ) 26

110 . 26

111 ] 27

112 t 28

113 i 28

114 : 28

115 l 28

116 . 28

117 t 29

118 i 29

119 : 29

120 l 29

121 . 29

122 t 30

123 i 30

124 : 30

125 s 30

126 ( 30

127 i 30

128 , 30

129 i 30

130 ) 30

131 . 30

132 p 31

133 ( 31

134 i 31

135 ) 31

136 . 31

137 t 32

138 i 32

139 : 32

140 ( 32

141 ( 32

142 l 32

143 + 32

144 l 32

145 ) 32

146 / 32

147 l 32

148 ) 32

149 . 32

150 p 33

151 ( 33

152 i 33

153 ) 33

154 . 33

155 ] 34

Таблица лексем:

0. tfi(ti)

1. [

2. ti:l.

3. i:ii+.

4. ri.

5. ]

7. m

8. [

9. ti:l.

10. p(i).

11. ti:l.

12. ti:s(i).

13. p(i).

14. ti:i(l).

15. p(i).

16. ti:l~.

17. p(i).

18. ti:ll|.

19. ti:ll&.

20. w(i>i)

21. [

22. p(i).

23. ]

24. e

25. [

26. p(i).

27. ]

28. ti:l.

29. ti:l.

30. ti:s(i,i).

31. p(i).

32. ti:ll+l/$$$$.

33. p(i).

34. ]

**Приложение Г**

struct MfstState //состояние автомата для сохранения

{

short lenta\_position; //позиция на ленте

short nrule; //номер текущего правила

short nrulechain; //номер текущей цепочки

MFSTSTSTACK st; //стек автомата

MfstState();

MfstState(

short pposition, //позиция на ленте

MFSTSTSTACK pst, //стек автомата

short pnrulechain //номер текущей цепочки, текущего правила

);

MfstState(

short pposition, //позиция на ленте

MFSTSTSTACK pst, //стек автомата

short pnrule, //номер текущего правила

short pnrulechain); //номер текущей цепочки, текущего правила

};

struct Rl

{

wchar\_t rulefile[PARAMETER\_MAX\_SIZE];

std::ofstream\* stream;

};

static const Rl FIRSTINIT = { L"", NULL }; //для начальной инициализации ID

Rl getrul(wchar\_t rulefile[]); //сформировать структуру ID

void WriteData(Rl rl); //время создания

struct Mfst //магазинный автомат

{

enum RC\_STEP //код возврата функции step

{

NS\_OK, //найдено правило и цепочка, цепочка записана в стек

NS\_NORULE, //не найдено правило грамматики (ошибки в грамматике)

NS\_NORULECHAIN, //не наёдена подходящая цепочка правила (ошибка в исходном коде

NS\_ERROR, //неизвестный нетерминальный символ грамматики

TS\_OK, //текущий символ ленты == вершине стека, продвинулась лента, pop стека

TS\_NOK, //текущий символ ленты != вершине стека, восстановлено состояние

LENTA\_END, //текущая позиция ленты >= lenta\_size

SURPRISE //неожиданный код возврата (ошибка в step)

};

struct MfstDiagnosis //диагностика

{

short lenta\_position; //позиция на ленте

RC\_STEP rc\_step; //код завершения шага

short nrule; //номер правила

short nrule\_chain; //номер цепочки правила

MfstDiagnosis();

MfstDiagnosis( //диагностика

short plenta\_position, //позиция на ленте

RC\_STEP prc\_step, //код завершения шага

short pnrule, //номер правила

short pnrule\_chain //номер цепочки правила

);

} diagnosis[MFST\_DIAGN\_NUMBER]; //последние самые глубокие сообщения

GRBALPHABET\* lenta; //перекодированная(TS/NS) лента (из LEX)

short lenta\_position; //текущая позиция на ленте

short nrule; //номер текущего правила

short nrulechain; //номер текущей цепочки, текущего правила

short lenta\_size; //размер ленты

GRB::Greibach grebach; //грамматика Грейбах

LEX::LexTable lexTable; //результат работы лексического анализатора

MFSTSTSTACK st; //стек автомата

std::vector<MfstState> storestate; // стек для сохранения состояний

Mfst();

Mfst(

LEX::LexTable plex, //результат работы лексического анализатора

GRB::Greibach pgrebach, //грамматика Грейбах

MFST::Rl rl

);

MFST::Rl rl;

char\* getJSt(char\* buf); //получить содержимое стека

char\* getCLenta(char\* buf, short pos, short n = 25); //лента: n символов с pos

char\* getDiagnosis(short n, char\* buf); //получить n-ую цепочку диагностики или 0x00

bool savestate(); //сохранить состояние автомата

bool reststate(); //восстановить состояние автомата

bool push\_chain( //поместить цепочку правила в стек

GRB::Rule::Chain chain //цепочка правила

);

RC\_STEP step(); //выполнить шаг автомата

bool start(); //запустить автомат

bool savediagnosis(

RC\_STEP pprc\_step //код завершения шага

);

void printrules(); //вывести последовательность правил

struct Deducation //вывод

{

short size; //количество шагов в выводе

short\* nrules; //номер правил грамматики

short\* nrulechains; //номер цепочек правил (nrules)

Deducation() { size = 0; nrules = 0; nrulechains = 0; }

} deducation;

bool savededucation(); //сохранить дерево вывода

};

# **Приложение Д**

Начало разбора

Шаг: Правило Входная лента Стек

0 : S->tfi(Q)[Y]S tfi(ti)[ti:l.i:i+i.ri.]m[ S$

0 : SAVESTATE: 1

0 : tfi(ti)[ti:l.i:i+i.ri.]m[ tfi(Q)[Y]S$

1 : fi(ti)[ti:l.i:i+i.ri.]m[t fi(Q)[Y]S$

2 : i(ti)[ti:l.i:i+i.ri.]m[ti i(Q)[Y]S$

3 : (ti)[ti:l.i:i+i.ri.]m[ti: (Q)[Y]S$

4 : ti)[ti:l.i:i+i.ri.]m[ti:l Q)[Y]S$

5 : Q->ti ti)[ti:l.i:i+i.ri.]m[ti:l Q)[Y]S$

5 : SAVESTATE: 2

5 : ti)[ti:l.i:i+i.ri.]m[ti:l ti)[Y]S$

6 : i)[ti:l.i:i+i.ri.]m[ti:l. i)[Y]S$

7 : )[ti:l.i:i+i.ri.]m[ti:l.p )[Y]S$

8 : [ti:l.i:i+i.ri.]m[ti:l.p( [Y]S$

9 : ti:l.i:i+i.ri.]m[ti:l.p(i Y]S$

10 : Y->ti:E.Y ti:l.i:i+i.ri.]m[ti:l.p(i Y]S$

10 : SAVESTATE: 3

Конец разбора

419 : N->p(i).N p(i).] N]$

419 : SAVESTATE: 55

419 : p(i).] p(i).N]$

420 : (i).] (i).N]$

421 : i).] i).N]$

422 : ).] ).N]$

423 : .] .N]$

424 : ] N]$

425 : TNS\_NORULECHAIN/NS\_NORULE

425 : RESSTATE

425 : p(i).] N]$

426 : N->p(i). p(i).] N]$

426 : SAVESTATE: 55

426 : p(i).] p(i).]$

427 : (i).] (i).]$

428 : i).] i).]$

429 : ).] ).]$

430 : .] .]$

431 : ] ]$

432 : $

433 : LENTA\_END

434 : -------> NS\_LENTA\_END

0 : S->tfi(Q)[Y]S 10

4 : Q->ti 2

8 : Y->ti:E.Y 6

11 : E->l 1

13 : Y->i:P.Y 5

15 : P->iMP 3

16 : M->+ 1

17 : P->i 1

19 : Y->ri. 3

23 : S->m[N] 4

25 : N->ti:E.N 6

28 : E->l 1

30 : N->p(i).N 6

35 : N->ti:E.N 6

38 : E->l 1

40 : N->ti:E.N 6

43 : E->s(i) 4

48 : N->p(i).N 6

53 : N->ti:P.N 6

56 : P->iP 2

57 : P->(P) 3

58 : P->l 1

61 : N->p(i).N 6....

66 : N->ti:E.N 6

69 : E->~l 2..

72 : N->p(i).N 6...

77 : N->ti:P.N 6...

80 : P->lMP 3..

81 : M->| 1..

82 : P->l 1..

84 : N->ti:P.N 6..

87 : P->lMP 3..

88 : M->& 1

89 : P->l 1

91 : N->w(K)[N]e[N]N 12

93 : K->iBi 3///

94 : B->> 1

98 : N->p(i). 5

106 : N->p(i). 5

112 : N->ti:E.N 6///

115 : E->l 1..

117 : N->ti:E.N 6

120 : E->l 1

122 : N->ti:E.N 6...

125 : E->s(i,i) 6...

132 : N->p(i).N 6...

137 : N->ti:P.N 6

140 : P->(P) 3

141 : P->(P)MP 5

142 : P->lMP 3

143 : M->+ 1

144 : P->l 1

146 : M->/ 1

147 : P->l 1

150 : N->p(i). 5

# **Приложение Е**

Таблица лексем:

0. tfi(ti)

1. [

2. ti:l.

3. i:ii+.

4. ri.

5. ]

7. m

8. [

9. ti:l.

10. p(i).

11. ti:l.

12. ti:s(i).

13. p(i).

14. ti:i(l).

15. p(i).

16. ti:l~.

17. p(i).

18. ti:ll|.

19. ti:ll&.

20. w(i>i)

21. [

22. p(i).

23. ]

24. e

25. [

26. p(i).

27. ]

28. ti:l.

29. ti:l.

30. ti:s(i,i).

31. p(i).

32. ti:ll+l/$$$$.

33. p(i).

34. ]

**Приложение Ж**

function myuf(a)

{

var t = 0o33;

if(t<0 )

{alert('because of minus t becomes to zero');

t = 0;}

if(!Number.isInteger(t)){alert('not an Integer, t will be rouned'); t=Math.round(j);}

t = t + a ;

if(t<0 )

{alert('because of minus t becomes to zero');

t = 0;}

if(!Number.isInteger(t)){alert('not an Integer, t will be rouned'); t=Math.round(j);}

return t ;

}

{

var check = true;

document.write('<br>');

document.write(check);

var str = "hello";

var sss = str.slice(0);

document.write('<br>');

document.write(sss);

var m = myuf(3);

if(m<0 )

{alert('because of minus m becomes to zero');

m = 0;}

if(!Number.isInteger(m)){alert('not an Integer, m will be rouned'); m=Math.round(j);}

document.write('<br>');

document.write(m);

var inverse = ~ 15;

if(inverse<0 )

{alert('because of minus inverse becomes to zero');

inverse = 0;}

if(!Number.isInteger(inverse)){alert('not an Integer, inverse will be rouned'); inverse=Math.round(j);}

document.write('<br>');

document.write(inverse);

var or = 55 | 80;

if(or<0 )

{alert('because of minus or becomes to zero');

or = 0;}

if(!Number.isInteger(or)){alert('not an Integer, or will be rouned'); or=Math.round(j);}

var and = 56 & 76;

if(and<0 )

{alert('because of minus and becomes to zero');

and = 0;}

if(!Number.isInteger(and)){alert('not an Integer, and will be rouned'); and=Math.round(j);}

if (or > and)

{

document.write('<br>');

document.write(or);

}

else

{

document.write('<br>');

document.write(and);

}

var n = "hello";

var s = "friend";

var ns = n + s;

document.write('<br>');

document.write(ns);

var j = ((44 + 4) / 7);

if(j<0 )

{alert('because of minus j becomes to zero');

j = 0;}

if(!Number.isInteger(j)){alert('not an Integer, j will be rouned'); j=Math.round(j);}

document.write('<br>');

document.write(j);

}