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 ADS-2018»

Выполнил студент Агапкина Диана Сергеевна

(Ф.И.О.)

Руководитель проекта преп. ст. Рауба Алексей Александрович

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

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

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

Консультанты ст.пр. Наркевич Аделина Сергеевна

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

преп. ст. Рауба Алексей Александрович

(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 2018

Оглавление

[Введение 5](#_Toc532670046)

[1 Спецификация языка программирования 6](#_Toc532670047)

[1.1 Характеристика языка программирования 6](#_Toc532670048)

[1.2 Алфавит языка 6](#_Toc532670049)

[1.3 Символы сепараторы 7](#_Toc532670050)

[1.4 Применяемые кодировки 7](#_Toc532670051)

[1.5 Типы данных 7](#_Toc532670052)

[1.6 Преобразование типов данных 8](#_Toc532670053)

[1.7 Идентификаторы 8](#_Toc532670054)

[1.8 Литералы 8](#_Toc532670055)

[1.9 Область видимости идентификаторов 8](#_Toc532670056)

[1.10 Инициализация данных 9](#_Toc532670057)

[1.11 Инструкции языка 9](#_Toc532670058)

[1.12 Операции языка 10](#_Toc532670059)

[1.13 Выражения и их вычисления 10](#_Toc532670060)

[1.14 Программные конструкции языка 10](#_Toc532670061)

[1.15 Область видимости 11](#_Toc532670062)

[1.16 Семантические проверки 11](#_Toc532670063)

[1.17 Распределение оперативной памяти на этапе выполнения 11](#_Toc532670064)

[1.18 Стандартная библиотека и её состав 12](#_Toc532670065)

[1.19 Ввод и вывод данных 12](#_Toc532670066)

[1.20 Точка входа 12](#_Toc532670067)

[1.21 Препроцессор 12](#_Toc532670068)

[1.22 Соглашения о вызовах 13](#_Toc532670069)

[1.23 Объектный код 13](#_Toc532670070)

[1.24 Классификация сообщений транслятора 13](#_Toc532670071)

[1.25 Контрольный пример 13](#_Toc532670072)

[2 Структура транслятора 14](#_Toc532670073)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 14](#_Toc532670074)

[2.2 Перечень входных параметров транслятора 15](#_Toc532670075)

[2.3 Перечень протоколов, формируемых транслятором и их содержимое 15](#_Toc532670076)

[3 Разработка лексического анализатора 16](#_Toc532670077)

[3.1 Структура лексического анализатора 16](#_Toc532670078)

[3.2 Контроль входных символов 16](#_Toc532670079)

[3.3 Удаление избыточных символов 17](#_Toc532670080)

[3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов 17](#_Toc532670081)

[3.5 Основные структуры данных 18](#_Toc532670082)

[3.6 Принцип обработки ошибок 18](#_Toc532670083)

[3.7 Структура и перечень сообщений лексического анализатора 18](#_Toc532670084)

[3.8 Параметры лексического анализатора и режимы его работы 19](#_Toc532670085)

[3.9 Алгоритм лексического анализа 19](#_Toc532670086)

[3.10 Контрольный пример 20](#_Toc532670087)

[4 Разработка синтаксического анализатора 21](#_Toc532670088)

[4.1 Структура синтаксического анализатора 21](#_Toc532670089)

[4.2 Контекстно свободная грамматика, описывающая синтаксис языка 21](#_Toc532670090)

[4.3 Построение конечного магазинного автомата 23](#_Toc532670091)

[4.4 Основные структуры данных 24](#_Toc532670092)

[4.5 Описание алгоритма синтаксического разбора 24](#_Toc532670093)

[4.6 Структура и перечень сообщений синтаксического анализатора 24](#_Toc532670094)

[4.7 Параметры синтаксического анализатора и режимы его работы 25](#_Toc532670095)

[4.8 Принцип обработки ошибок 25](#_Toc532670096)

[4.9 Контрольный пример 25](#_Toc532670097)

[5 Разработка семантического анализатора 26](#_Toc532670098)

[5.1 Структура семантического анализатора 26](#_Toc532670099)

[5.2 Функции семантического анализатора 26](#_Toc532670100)

[5.3 Структура и перечень сообщений семантического анализатора 26](#_Toc532670101)

[5.4 Принцип обработки ошибок 26](#_Toc532670102)

[5.5 Контрольный пример 26](#_Toc532670103)

[6 Преобразование выражений 27](#_Toc532670104)

[6.1 Выражения, допускаемые языком 27](#_Toc532670105)

[6.2 Польская запись 27](#_Toc532670106)

[6.3 Программная реализация обработки выражений 28](#_Toc532670107)

[6.4 Контрольный пример 28](#_Toc532670108)

[7 Генерация кода 29](#_Toc532670109)

[7.1 Структура генератора кода 29](#_Toc532670110)

[7.2 Представление типов данных в оперативной памяти 29](#_Toc532670111)

[7.3 Алгоритм работы генератора кода 30](#_Toc532670112)

[8 Тестирование транслятора 31](#_Toc532670113)

[8.1 Тестирование фазы проверки на допустимость символов 31](#_Toc532670114)

[8.2 Тестирование лексического анализатора 31](#_Toc532670115)

[8.3 Тестирование синтаксического анализатора 31](#_Toc532670116)

[8.4 Тестирование семантического анализатора 31](#_Toc532670117)

[Заключение 32](#_Toc532670118)

[Список используемых источников 33](#_Toc532670119)

[Приложениe A 34](#_Toc532670120)

[Приложение Б 43](#_Toc532670121)

[Приложение В 45](#_Toc532670122)

[Приложение Г 47](#_Toc532670123)

[Приложение Д 50](#_Toc532670124)

[Приложение Е 54](#_Toc532670125)

# **Введение**

Главной целью данной курсовой работы является разработка компилятора для языка программирования ADS-2018. Основная задача компилятора заключается в том, чтобы сделать программу, написанную языке программирования ADS-2018, понятной компьютеру. Этого можно добиться одним из двух способов: компиляцией или интерпретацией. В данном курсовом проекте трансляция будет осуществляться в код на языке MASM.

Мой компилятор состоит из следующих составных частей:

– лексический и семантический анализаторы;

– синтаксический анализатор;

– генератор исходного кода на языке ассемблера.

Исходя из цели курсового проекта, были определены следующие задачи:

– разбработка спецификации языка программирования;

– разбратка структуры транслятора;

– разработка лексического анализатора;

– разработка синтаксического анализатора;

– разработка семантического анализатора;

– обработка выражений;

– генерация кода на язык MASM;

– тестирование транслятора.

Язык программирования ADS-2018 предназначен для выполнения простейших арифметический действий и операций над строками.

# **1 Спецификация языка программирования**

## **Характеристика языка программирования**

Разработанный язык является универсальным процедурным языком. Преобразование типов не допускается, из этого следует, что ADS-2018 – строго типизированный язык программирования. Вся объектно-ориентированная часть языков программирования исключена и не используется в данном языке, из чего следует, что он не является объектно-ориентированным. Язык является транслируемым.

* 1. **Алфавит языка**

Алфавит языка ADS-2018 основан на кодировке Windows-1251, представленной на рисунке 1.1.

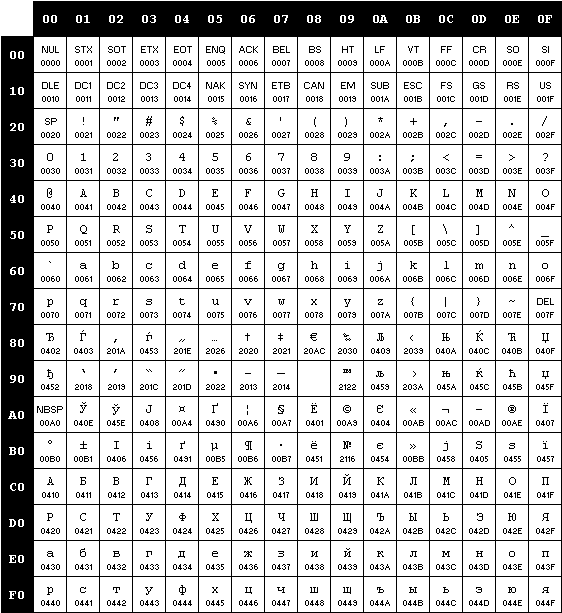


Рисунок 1.1 – Алфавит входных символов

Исходный код ADS-2018 может содержать символы латинского алфавита, цифры десятичной системы счисления от 0 до 9, русские символы разрешены только в строковых литералах.

## **Символы сепараторы**

Символы, которые являются сепараторами представлены в таблице 1.1.

Таблица 1.1 – Символы-сепараторы

|  |  |
| --- | --- |
| Символ(ы) | Назначение |
| = | Оператор присваивания |
| > < | Логические операции (операции сравнения: больше/меньше). Используются в условных конструкциях |
| , | Разделитель параметров функции |
| ‘пробел’ | Разделитель цепочек. Допускается везде, кроме названий идентификаторов и ключевых слов |
| + - \* | Арифметические операции |
| {  } | Блок функции  Блок, отделяющий условные конструкции |
| (  ) | Параметры функции/приоритетность операций (в выражениях) |

## **Применяемые кодировки**

Для написания исходного кода на языке программирования ADS-2018 используется кодировка Windows-1251.

## **Типы данных**

В языке ADS-2018 есть 2 типа данных: целочисленный и строковый. Описание типов данных, предусмотренных в данным языке представлено в таблице 1.2.

Таблица 1.2 – Типы данных языка ADS-2018

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| integer | Фундаментальный тип данных. Предусмотрен для объявления целочисленных положительных данных (4 байта).  Автоматически инициализируется нулевым значением.  Поддерживаемые операции:  **+** (бинарный) – оператор сложения;  - (бинарный) – оператор вычитания;  **\*** (бинарный) – оператор умножения;  **=** (бинарный) – оператор присваивания.  В качестве условия цикла/условного оператора поддерживаются следующие логические операции:  **>** (бинарный) – оператор «больше»;  **<** (бинарный) – оператор «меньше»; |
| string | Фундаментальный тип данных. Предусмотрен для объявления строк. (1 символ – 1 байт). Автоматическая инициализация строкой нулевой длины. Максимальное количество символов в строке – 255.  Операции над данными строкового типа: возможно присваивание строковому идентификатору значения другого строкового идентификатора, строкового литерала или значения строковой функции. |

## **Преобразование типов данных**

В языке программирования ADS-2018 преобразование типов данных не предусмотрено по причине того, что язык является строго типизированным. Попытка преобразования типов приведет к семантической ошибке.

## **Идентификаторы**

Общее количество идентификаторов ограничено только максимальным размером таблицы идентификаторов. Идентификаторы представляют собой произвольно длинную последовательность символов нижнего регистра латинского алфавита ([a..z]). Однако, длина идентификатора на этапе лексического разбора усекается до 8 символов. Данные правила действуют для всех типов идентификаторов. Зарезервированные идентификаторы не предусмотрены.

Идентификаторы не должны совпадать с ключевыми словами языка ADS-2018. Типы идентификаторов: имя переменной, имя функции, параметр функции, литерал, арифметический оператор.

Имена идентификаторов функций не должны совпадать с командами ассемблера.

## **Литералы**

С помощью литералов осуществляется инициализация переменных. Все литералы являются rvalue. Краткое описание литералов языка ADS-2018 представлено в таблице 1.3.

Таблица 1.3 – Описание литералов

|  |  |
| --- | --- |
| Тип литерала | Описание |
| Литералы целого типа  Integer | Последовательность цифр ([0..9]). Только неотрицательные значения. Инициализация значением 0 по умолчанию. |
| Строковые литералы  String | Набор символов, заключённых в “” (двойные кавычки). Инициализируются пустой строкой по умолчанию.  Внутри литерала не допускается использование одинарных и двойных кавычек. |

## **Область видимости идентификаторов**

Для объявления переменной используется ключевое слово new, после которого указывается тип данных и имя идентификатора. Допускается инициализация при объявлении.

Пример объявления переменной числового типа с инициализацией:

new integer num = 10;

Пример объявления переменной строкового типа с инициализацией:

new string str = “Hello World”;

Область видимости «сверху вниз» (по принципу С++). В языке ADS-2018 требуется обязательное объявление переменной перед её использованием. Все переменные должны находиться внутри программного блока языка. Имеется возможность объявления одинаковых переменных в разных блоках. Каждая переменная или параметр функции получают префикс – название функции, в которой они находятся.

Для объявления функции используется ключевое слово function, перед которым указывается тип функции, а после – идентификатор (имя функции). Далее обязателен список параметров и тело функции.

## **Инициализация данных**

Таблица 1.4 – Способы инициализации переменных

|  |  |
| --- | --- |
| Вид инициализации | Примечание |
| new <тип данных><идентификатор>; | Автоматическая инициализация: переменные типа integer инициализируются нулём, переменные типа string – пустой строкой. |
| <идентификатор> = <значение>; | Присваивание переменной значения. |

## **Инструкции языка**

Все возможные инструкции языка программирования ADS-2018 представлены в общем виде в таблице 1.5.

Таблица 1.5 – Инструкции языка программирования ADS-2018

|  |  |
| --- | --- |
| Инструкция | Запись на языке ADS-2018 |
| Объявление переменной | new <тип данных> <идентификатор>; |
| Присваивание | new <тип данных> <идентификатор> = <значение>/<идентификатор>; |
| Объявление внешней функции | <тип данных> function <идентификатор> (<тип данных> <идентификатор>, …) {…} |
| Условная конструкция | if (<идентификатор>/<литерал> >/< <идентификатор>/<литерал>) then {  …  }  else {  …  } |
| Возврат из подпрограммы | ret <идентификатор> / <литерал>; |
| Вывод данных | print (<идентификатор> / <литерал>); |

## **Операции языка**

Язык программирования ADS-2018 может выполнять арифметические операции, представленные в таблице 1.6.

Таблица 1.6 – Приоритетности операций языка программирования ADS-2018

|  |  |
| --- | --- |
| Операция | Приоритетность операции |
| ( ) | 0 или 4 |
| , | 1 |
| \* | 2 |
| + - | 3 |

Максимальным значением приоритетности является “0”, минимальным “4” соответственно.

Также языком поддерживается операция сравнения операндов. Данная операция не имеет приоритета.

## **Выражения и их вычисления**

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

1. Допускается использовать скобки для смены приоритета операций;
2. Выражение записывается в строку без переносов;
3. Использование двух подряд идущих операторов не допускается;
4. Допускается использовать в выражении вызов функции, вычисляющей и возвращающей целочисленное значение.

## **Программные конструкции языка**

Программа на языке ADS-2018 оформляется в виде функций пользователя и главной функции. При составлении функций требуется выделять блоки и фрагменты, применять отступы и т.д.

Ключевые программные конструкции языка программирования ADS-2018 представлены в таблице 1.7.

Таблица 1.7 – Программные конструкции языка ADS-2018

|  |  |
| --- | --- |
| Главная функция (точка входа в приложение) | main{  …  } |
| Функция | <тип> function <идентификатор> (<тип> <идентификатор>, …) {  …  ret <выражение>;  } |
| Условная конструкция | if (<идентификатор>/<литерал> >/< <идентификатор>/<литерал>) then {  …  }  else {  …  }  <ид1>, <ид2> - идентификаторы или литералы целочисленного типа. <оператор> - один из операторов сравнения ( **> <**), устанавливающий отношение между двумя операндами и организующий условие данной конструкции. При истинности условия выполняется код внутри блока if, иначе – внутри блока else. Любой из блоков if, else может полностью отсутствовать, но не оба блока одновременно. При отсутствии одного из блоков, в зависимости от истинности или ложности условия программа может как выполнить один из заявленных блоков, так и передать управление инструкции, следующей в коде за закрывающим условную конструкцию символом **‘}’**. |

## **Область видимости**

В языке ADS-2018 переменные являются локальными и обязаны находиться внутри программного блока функций (по принципу С++). Объявление глобальных переменных не предусмотрено.

## **Семантические проверки**

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

Таблица 1.8 – Семантические проверки

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Идентификаторы функций не должны повторяться |
| 2 | Тип данных передаваемых значений в функцию должен совпадать с типом параметров при её объявлении |
| 3 | Тип данных передаваемых значений в функцию стандартной библиотеки должен соответствовать заявленному. |
| 4 | Идентификатор должен быть объявлен до его использования. |
| 5 | Операнды в арифметическом выражении не могут быть разных типов |

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

Все переменные размещаются в куче. Распределение оперативной памяти происходит на этапе генерации. Промежуточный код, таблица лексем и таблица идентификаторов сохраняются в структуры с выделенной под них динамической памятью, которая очищается по окончанию работы транслятора.

## **Стандартная библиотека и её состав**

В языке ADS-2018 присутствует стандартная библиотека, которая подключается автоматически при трансляции исходного кода в язык ассемблера.

Стандартная библиотека написанная на языке С++, подключается к транслированному коду на этапе генерации кода.

Вызовы стандартных функций доступны там же, где и вызов пользовательских функций. Также в стандартной библиотеке реализованы функции для манипулирования вводом-выводом, недоступные конечному пользователю.

Содержимое библиотеки и описание содержащихся в ней функций представлено в таблице 1.9

Таблица 1.9 – Состав стандартной библиотеки

|  |  |  |  |
| --- | --- | --- | --- |
| Имя функции | Возвращаемое значение | Принимаемые параметры | Описание |
| pows | integer | integer x – число  integer n – степень | Функция возводит число x в степень n |
| len | integer | string x - строка | Функция вычисляет длину строки x |
| printi | 0 | integer x - число | Функция выводит на консоль число x |
| prints | 0 | string x - строка | Функция выводит на консоль строку x |

## **Ввод и вывод данных**

Ввод данных не поддерживается языком программирования ADS-2018.

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

Функция, управляющая выводом данных, реализована на языке С++, вызывается из транслированного кода, конечному пользователю недоступны. Пользовательская команда print в транслированном коде будет заменена вызовом нужных библиотечных функций. Библиотека, содержащая нужные процедуры, подключается на этапе генерации кода.

## **Точка входа**

В языке ADS-2018 каждая программа должна содержать главную функцию (точку входа) main, с первой инструкции которой начнётся последовательное выполнение команд программы.

## **Препроцессор**

Препроцессор, принимающий и выдающий некоторые данные на вход транслятору, в языке программирования ADS-2018 предусмотрен.

## **Соглашения о вызовах**

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

– все параметры функции передаются через стек;

– память высвобождает вызываемый код;

– занесение в стек параметров идёт справа налево.

## **Объектный код**

ADS-2018 транслируется в язык ассемблера.

## **Классификация сообщений транслятора**

Генерируемые транслятором сообщения определяют степень его информативности, то есть сообщения транслятора должны давать максимально полную информацию о допущенной пользователем ошибке при написании программы на языке ADS-2018. Сообщения транслятора приведены в таблице 1.10.

Таблица 1.10 – Классификация сообщений транслятора

|  |  |
| --- | --- |
| Интервал | Описание ошибок |
| 0-99 | Системные ошибки |
| 100-109 | Ошибки параметров |
| 110-119 | Ошибки открытия и чтения файлов |
| 120-129 | Ошибки лексического анализа |
| 200-299 | Ошибки синтаксического анализа |
| 300-309 | Ошибки семантического анализа |

## **Контрольный пример**

Контрольный пример представлен в Приложении А.

# **2 Структура транслятора**

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

Транслятор преобразует программу, написанную на языке ADS-2018 в программу на языке ассемблера. Компонентами транслятора являются лексический, синтаксический и семантический анализаторы, а также генератор кода на язык ассемблера. Принцип их взаимодействия представлен на рисунке 2.1.



Рисунок 2.1 – Структура транслятора

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

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

В моём трансляторе этапы лексического и семантического анализа являются единым целым и выполняются одновременно.

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

Генератор кода – этап транслятора, выполняющий генерацию ассемблерного кода на основе полученных данных на предыдущих этапах трансляции. Генератор кода принимает на вход таблицы идентификаторов и лексем и транслирует код на языке ADS-2018, прошедший все предыдущие этапы, в код на языке Ассемблера. Более полно описан в главе 7.

## **2.2 Перечень входных параметров транслятора**

Входные параметры представлены в таблице 2.1.

Таблица 2.1 – Входные параметры транслятора языка ADS-2018

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<имя\_файла> | Входной файл с расширением .txt, в котором содержится исходный код на ADS-2018 | Не предусмотрено |
| -log:<имя\_файла> | Файл для записи результата проверки входного файла на допустимость символов | <имя\_файла>.log |
| -out:<имя\_файла> | Файл для записи результата работы транслятора | <имя\_файла>.asm |
| -lex:<имя\_файла> | Файл для записи результата работы лексического и семантического анализа. | <имя\_файла>.lex |
| -syn:<имя\_файла> | Файл для записи результата работы синтаксического анализа. | <имя\_файла>.syn |

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

Таблица с перечнем протоколов, формируемых транслятором языка

ADS-2018 и их назначением представлена в таблице 2.2

Таблица 2.2 – Протоколы, формируемые транслятором языка ADS-2018

|  |  |
| --- | --- |
| Формируемый протокол | Описание протокола |
| Файл журнала с параметром <log> | Содержит информацию о входных параметрах в приложение и о этапе проверки символов на допустимость. |
| Выходной файл c параметром <out> | Содержит сгенерированный код на языке Ассемблера. |
| lex:<имя\_файла> | Результат работы лексического и семантического анализа. Содержит таблицы лексем и идентификаторов. |
| syn:<имя\_файла> | Результат работы синтаксического анализа. Содержит правила разбора, трассировку, а также преобразованные после польской записи таблицы лексем и идентификаторов. |

# **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 ADS-2018. На выходе формируется таблица лексем и таблица идентификаторов. Структура лексического анализатора представлена на рисунке 3.1



Рисунок 3.1 – Структура лексического анализатора ADS-2018

## **3.2 Контроль входных символов**

Таблица для контроля входных символов представлена на рисунке 3.2

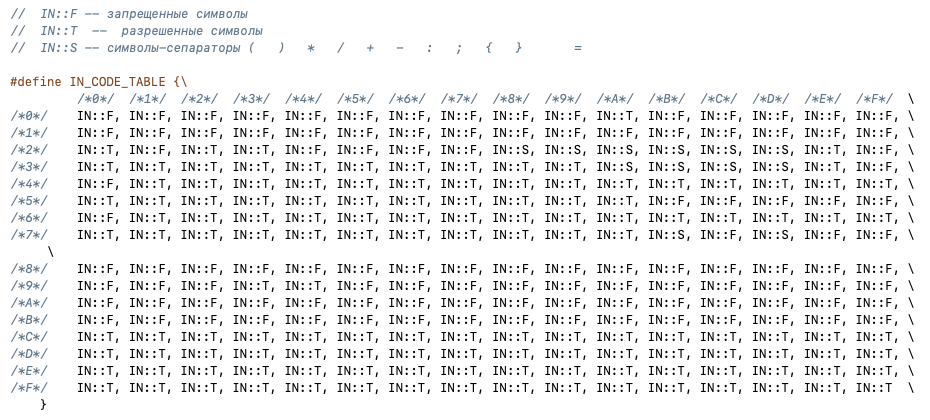


Рисунок 3.2 – Таблица контроля входных символов

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, S – сепаратор.

## **3.3 Удаление избыточных символов**

Избыточными символами являются символы табуляции и пробелы.

Избыточные символы удаляются на этапе разбиения исходного кода на токены.

Описание алгоритма удаления избыточных символов, а также разбиения исходного кода на цепочки:

1. Посимвольно считываем файл с исходным кодом программы;

2. Встреча пробела или знака табуляции включает проверку предыдущего (буферного) символа;

2.1. Если предыдущий символ является пробелом, то текущий символ игнорируется;

2.2. Если предыдущий символ является допустимым символом, то текущий пробел – разделитель единиц языка, следовательно, сохраняем считанную единицу в формируемую структуру первично обработанного исходного кода;

3. В отличие от других символов-сепараторов не записываем в очередь лексем эти символы, т.е. игнорируем.

## **3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов**

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

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

|  |  |  |
| --- | --- | --- |
| Тип цепочки | Цепочка | Лексема |
| Ключевые слова | new | n |
| integer | x |
| string | s |
| function | f |
| print | o |
| pows | p |
| len | c |
| ret | r |
| main | m |
| Иное | Идентификатор | i |
| Литерал | l |
| Тип данных | t |

Продолжение таблицы 3.1

|  |  |  |
| --- | --- | --- |
| Сепараторы | ; | ; |
| , | , |
| { | { |
| } | } |
| ( | ( |
| ) | ) |
| = | = |
| > | > |
| < | < |
| Операторы | - | a |
| \* | a |
| + | a |

Пример реализации таблицы лексем представлен в приложении А.

Также в приложении А находятся конечные автоматы, соответствующие лексемам языка ADS-2018.

## **3.5 Основные структуры данных**

Основные структуры таблиц лексем и идентификаторов данных языка ADS-2018, используемых для хранения, представлены в приложении А. В таблице лексем содержится лексема, её номер, полученный при разборе, номер строки в исходном коде и приоритет. В таблице идентификаторов содержится имя идентификатора, номер в таблице лексем, тип данных, смысловой тип идентификатора и его значение.

## **3.6 Принцип обработки ошибок**

В трансляторе ADS-2018 не предусмотрена обработка сразу нескольких ошибок, а также отсутствуют предупреждения.

При возникновении критической ошибки – работа транслятора прекращается. Вызывается функция получения ошибки, в которую передается, в зависимости от места возникновения ошибки, следующая информация: код ошибки и сообщение, номер строки в коде, номер позиции в строке или только код ошибки.

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

Перечень сообщений, формируемых лексическим анализатором в ходе своей работы, представлен на рисунке 3.3.

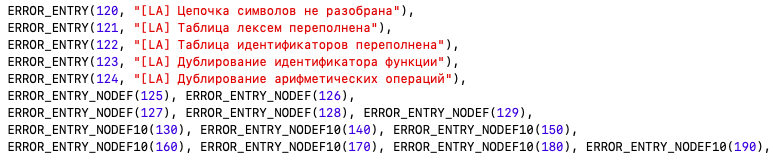


Рисунок 3.3 – Перечень ошибок лексического анализатора

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

Входным параметром лексического анализа является очередь, состоящая из структур, полями которых являются лексема и номер её строки в исходном файле, полученные на этапе проверки исходного кода на допустимость символов.

Транслятор ADS-2018 допускает использование параметров для управления работой лексического анализатора, а именно: выводом таблицы лексем и таблицы идентификаторов. Описание параметров представлено в таблице 2.1.

## **3.9 Алгоритм лексического анализа**

Лексический анализ выполняется программой (входящей в состав транслятора), называемой лексическим анализатором. Цель лексического анализа — выделение и классификация лексем в тексте исходной программы. Лексический анализатор распознаёт и разбирает цепочки исходного текста программы. 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.

Пример. Регулярное выражение для ключевого слова real: ‘new’.

Граф конечного автомата для этой лексемы представлен на рисунке 3.4. S0 – начальное состояние, S4 – конечное состояние автомата.

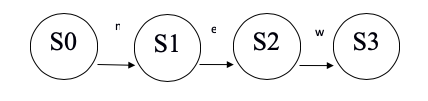


Рисунок 3.4 – Граф переходов для цепочки ‘new’

## **3.10 Контрольный пример**

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

# **4 Разработка синтаксического анализатора**

## **4.1 Структура синтаксического анализатора**

Синтаксический анализ – это фаза трансляции, выполняемая после лексического анализа и предназначенная для распознавания синтаксических конструкций. Входом для синтаксического анализа является таблица лексем и таблица идентификаторов, полученные после фазы лексического анализа. Выходом – дерево разбора. Структура синтаксического анализатора представлена на рисунке 4.1.



Рисунок 4.1 – Структура синтаксического анализатора

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

В синтаксическом анализаторе транслятора языка ADS-2018 используется контекстно-свободная грамматика , где

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

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

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

S – начальный символ грамматики, являющийся нетерминалом.

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

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

Грамматика языка ADS-2018 представлена в приложении Б.

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

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

Таблица 4.1 – Перечень правил, составляющих грамматику языка и описание нетерминальных символов ADS-2018.

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| S | tfi(F){N}S  m{N} | Порождает правила, описывающее общую структуру программы |
| N | nti;N  nti=E;N  rE;  i=E;N  o(i);N  o(s);N  o(x);N  o(s);  o(i);  o(x);  ?Q1{N}2{N}N  ?Q1{N}N | Порождает правила, описывающие инструкции языка  Порождает правила, описывающие параметры стандартной функции print |
| E | i  s  x  (E)  xM  i(W)  iM  nM  (E)M  i(W)M  p(i,x)  p(x,x)  p(i,i)  p(x,i)  c(i)  c(s) | Порождает правила, описывающие выражения  Порождает правила, описывающие параметры стандартной функции pows  Порождает правила, описывающие параметры стандартной функции len |
| F | ti  ti,F | Порождает правила, описывающие параметры локальной функции при её объявлении |
| W | i  s  x  i,W  s,W  x,W | Порождает правила, описывающие параметры локальной функции при её вызове |
| M | a  aE  aEM | Порождает правила, описывающие арифметические действия |
| Q | (R<R)  (R>R) | Порождает правила, описывающие операнды оператора ветвления |
| R | i  x | Порождает правила, описывающие параметр оператора выхода из функции |

## **4.3 Построение конечного магазинного автомата**

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

Таблица 4.2 – Описание компонентов магазинного автомата

|  |  |  |
| --- | --- | --- |
| Компонента | Определение | Описание |
|  | Множество состояний автомата | Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата |
|  | Алфавит входных символов | Алфавит является множеством терминальных и нетерминальных символов, описание которых содержится в разделе 1.2 и в таблице 4.1. |
|  | Алфавит специальных магазинных символов | Алфавит магазинных символов содержит стартовый символ и маркер дна стека |
|  | Функция переходов автомата | Функция представляет из себя множество правил грамматики, описанных в таблице 4.1. |
|  | Начальное состояние автомата | Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики (нетерминальный символ S) |
|  | Начальное состояние магазина автомата | Символ маркера дна стека ($) |
|  | Множество конечных состояний | Конечные состояние заставляют автомат прекратить свою работу. Конечным состоянием является пустой магазин автомата и совпадение позиции на входной ленте автомата с размером ленты |

## **4.4 Основные структуры данных**

Основные структуры данных синтаксического анализатора включают в себя структуру магазинного автомата и структуру грамматики Грейбах, описывающей правила языка ADS-2018. Данные структуры представлены в приложении В.

## **4.5 Описание алгоритма синтаксического разбора**

Принцип работы автомата следующий:

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

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

Перечень сообщений синтаксического анализатора представлен на рисунке 4.2.

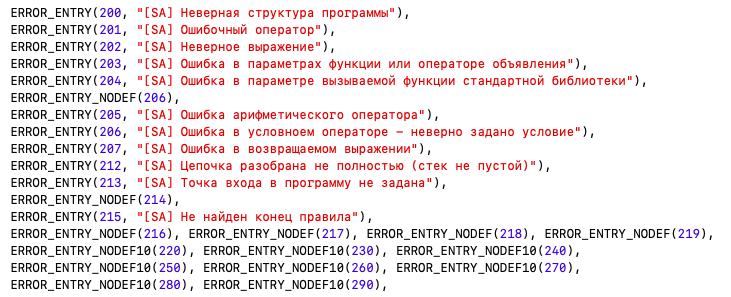


Рисунок 4.2 – Перечень сообщений синтаксического анализатора

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

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

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

## **4.8 Принцип обработки ошибок**

Обработка ошибок происходит следующим образом:

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

## **4.9 Контрольный пример**

Пример разбора синтаксическим анализатором исходного кода на языке ADS-2018 представлен в приложении Г. Дерево разбора исходного кода также представлено в приложении Г.

# **5 Разработка семантического анализатора**

## **5.1 Структура семантического анализатора**

Семантический анализ происходит при выполнении фазы лексического анализа и реализуется в виде отдельных проверок текущих ситуаций в конкретных случаях: установки флага или нахождении в особом месте программы (оператор выхода из функции, оператор ветвления, вызов функции стандартной библиотеки).

## **5.2 Функции семантического анализатора**

Семантический анализатор предназначен для проверки соответствия исходного кода спецификации, например, соответствие исходного кода заданной грамматике (синтаксический анализ), длина идентификаторов.

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

Все обнаруженные ошибки записываются в структуру. Сообщения, формируемые семантическим анализатором, представлены на рисунке 5.1.

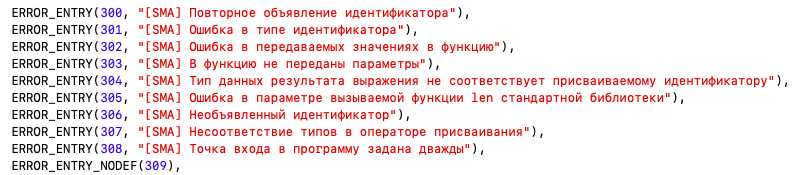


Рисунок 5.1 – Перечень сообщений семантического анализатора

## **5.4 Принцип обработки ошибок**

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

## **5.5 Контрольный пример**

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

## **6 Преобразование выражений**

## **6.1 Выражения, допускаемые языком**

В языке ADS-2018 допускаются выражения, применимые к целочисленным типам данных. В выражениях поддерживаются арифметические операции, такие как +, -, \* и (), и вызовы функций как операнды арифметических выражений.

Приоритет операций представлен в таблице 6.1.

Таблица 6.1 – Приоритет операций в языке ADS-2018

|  |  |
| --- | --- |
| Приоритет | Операция |
| 0 | ( |
| 0 | ) |
| 1 | , |
| 2 | + |
| 2 | - |
| 3 | \* |
| 4 | ( – скобка параметров функции |
| 4 | ) – скобка параметров функции |

## **6.2 Польская запись**

Выражения в языке ADS-2018 преобразовываются к обратной польской записи.

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

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

Алгоритм построения:

– исходная строка: выражение;

– результирующая строка: польская запись;

– стек: пустой;

– исходная строка просматривается слева направо;

– операнды переносятся в результирующую строку в порядке их следования;

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

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

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

– отрывающая скобка помещается в стек;

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

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

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

Таблица 6.2 – Пример преобразования выражения в обратную польскую запись

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

## **6.3 Программная реализация обработки выражений**

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

## **6.4 Контрольный пример**

Пример преобразования выражения к польской записи представлен в таблице 6.2. Преобразование выражений в формат польской записи необходимо для построения более простых алгоритмов их вычисления.

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

# **7 Генерация кода**

## **7.1 Структура генератора кода**

Генерация объектного кода — это перевод компилятором внутреннего представления исходной программы в цепочку символов выходного языка. На вход генератора подаются таблицы лексем и идентификаторов, на основе которых генерируется файл с ассемблерным кодом.

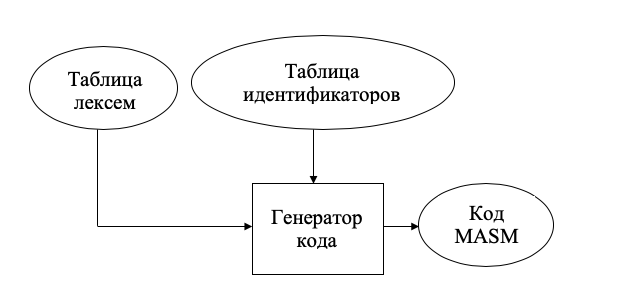


Рисунок 7.1 - Структура генератора кода

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

Элементы таблицы идентификаторов расположены в разных сегментах языка ассемблера – .data и .const. Идентификаторы языка ADS-2018 размещены в сегменте данных(.data). Литералы – в сегменте констант (.const). Соответствия между типами данных идентификаторов на языке ADS-2018 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка ADS-2018 и языка Ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке ADS-2018 | Тип идентификатора на языке ассемблера | Пояснение |
| integer | SDWORD | Хранит целочисленный тип данных со знаком. |
| string | DWORD | Хранит указатель на начало строки. |
| L(0-9) | BYTE  DWORD | Литералы: символьные,  целочисленные |

## **7.3 Алгоритм работы генератора кода**

Преобразования происходят по принципу, встретив определённую лексему и зная, в каком месте программы находится сейчас лексема, программа генерирует код на языке Ассемблера.

На рисунке 7.2 представлен пример описания лексемы на языке Ассемблера.

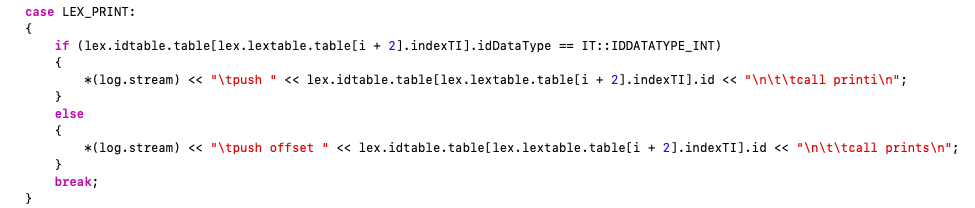


Рисунок 7.2 – Алгоритм для лексемы ‘o’

Генерируемый код записывается в файл, который был параметром ‑out. Сгенерированный код и результат программы можно посмотреть в приложении Е.

# **8 Тестирование транслятора**

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

В языке ADS-2018 не разрешается использовать запрещённые входным алфавитом символы. Результат использования запрещённого символа показан в таблице 8.1.

Таблица 8.1 – Тестирование фазы проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| ret &c; | Ошибка 111: Ошибка проверки входного файла. Недопустимый символ в исходном файле (-in), строка 3, позиция 3 |

## **8.2 Тестирование лексического анализатора**

На этапе лексического анализа могут возникнуть ошибки, описанные в пункте 3.2. Результаты тестирования лексического анализатора показаны в таблице 8.2.

Таблица 8.2 – Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| iтt funion one (int a, int b) | Ошибка 120: [LA] Цепочка символов не разобрана, строка 1 |

## **8.3 Тестирование синтаксического анализатора**

На этапе синтаксического анализа могут возникнуть ошибки, описанные в пункте 4.6. Результаты тестирования синтаксического анализатора показаны в таблице 8.3.

Таблица 8.3 – Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| e = (2\*17)) + 9 | Ошибка 207: [SA] Ошибка арифметического оператора строка 21 |

## **8.4 Тестирование семантического анализатора**

Итоги тестирования семантического анализатора приведены в таблице 8.4.

Таблица 8.4 – Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| new int e;  new int e; | Ошибка 300: [SMA] Повторное объявление идентификатора строка 18 |

# **Заключение**

В ходе выполнения курсовой работы был разработан компилятор для языка программирования ADS-2018. Таким образом, были выполнены основные задачи данной курсовой работы:

* Сформулирована спецификация языка ADS-2018;
* 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 компилятор с языка программирования ADS-2018 на язык MASM;
* Проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка ADS-2018 включает:

1. 2 типа данных;
2. поддержка операции вывода;
3. возможность вызова функций стандартной библиотеки;
4. наличие 3 арифметических операторов для вычисления выражений;
5. структурированная система для обработки ошибок пользователя.

Основные характеристики транслятора ADS-2018:

1. возможность обработки 4 входных параметров;
2. возможность обработки 33 ошибок;
3. реализация 29 конечных автоматов;
4. реализация 43 цепочек правил грамматики;
5. наличие порядка 3000 строк кода;

# **Список используемых источников**

1. Ахо, А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003. – 768с.
2. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. - 3-е изд. – Москва : Вильямс, 2003. - 429 с.
3. Страуструп, Б. Принципы и практика использования C++ / Б. Страуструп – 2009 – 1238 с.
4. \_\_stdcall [Электронный ресурс] – Режим доступа: <https://msdn.microsoft.com/en-us/library/zxk0tw93.aspx> - Дата доступа: 14.12.2016.

# **Приложениe A**

Листинг 1 - Контрольный пример

int function one (int a, int b) {

new int c;

c = (a+b)\*2;

ret c;

}

int function two (string d) {

new int e;

e = len(d);

ret e;

}

int function three (int x, int y) {

new int z;

z = pows(x, y);

ret z;

}

main {

new int e;

new int f = 59;

new int g;

new string result;

e = (2\*17) + 9;

g = one (e,f);

if (e < g) then {

print (e);

}

else {

print (g);

}

print (g);

result = "My name is Diana";

g = two (result);

print (g);

g = three (9,6);

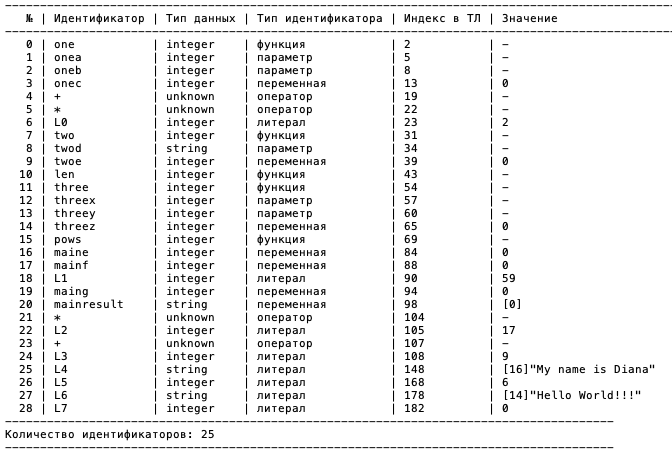
print (g);

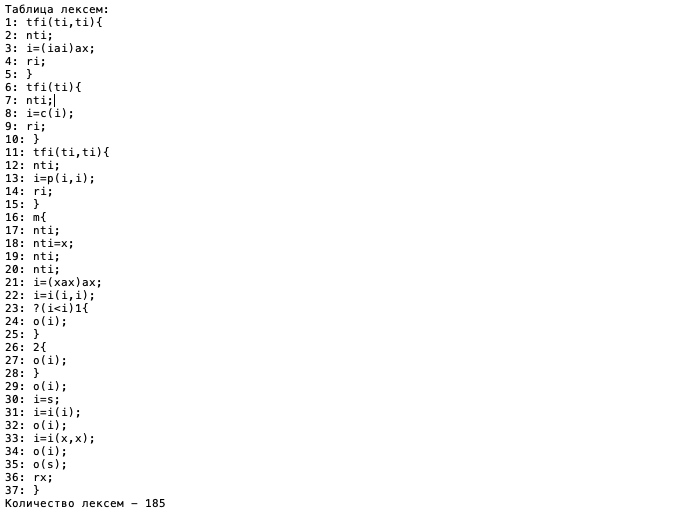
print ("Hello World!!!");

ret 0;

}

Листинг 2 – Таблица идентификаторов и лексем

****



Листинг 3 – Конечные автоматы, соответствующие лексемам языка

FST fstAriph(LEX\_ACTION, SYMB\_ACTION, "", 2,

NODE(3, RELATION('+', 1), RELATION('-', 1), RELATION('\*', 1)), \

NODE()

);

FST fstLeftHesis(LEX\_LEFTHESIS, SYMB\_NEED\_PRIORITY, "", 2,

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

NODE()

);

FST fstRightHesis(LEX\_RIGHTHESIS, SYMB\_NEED\_PRIORITY, "", 2,

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

NODE()

);

FST fstEqual(LEX\_EQUAL, SYMB\_LEXEMS, "", 2,

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

NODE()

);

FST fstLeftBrace(LEX\_LEFTBRACE, SYMB\_LEXEMS, "", 2,

NODE(1, RELATION('{', 1)), \

NODE()

);

FST fstRightBrace(LEX\_RIGHTBRACE, SYMB\_LEXEMS, "", 2,

NODE(1, RELATION('}', 1)), \

NODE()

);

FST fstLeftSqBrace(LEX\_LEFTSQBRACE, SYMB\_LEXEMS, "", 2,

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

NODE()

);

FST fstRightSqBrace(LEX\_RIGHTSQBRACE, SYMB\_LEXEMS, "", 2,

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

NODE()

);

FST fstSemicolon(LEX\_SEMICOLON, SYMB\_LEXEMS, "", 2,

NODE(1, RELATION(';', 1)), \

NODE()

);

FST fstMore(LEX\_MORE, SYMB\_LEXEMS, "", 2,

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

NODE()

);

FST fstLess(LEX\_LESS, SYMB\_LEXEMS, "", 2,

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

NODE()

);

FST fstComma(LEX\_COMMA, SYMB\_NEED\_PRIORITY, "", 2,

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

NODE()

);

FST fstNumLit(LEX\_INT, SYMB\_LITERAL, "", 2,

NODE(10,

RELATION('0', 1), RELATION('1', 1), RELATION('2', 1),

RELATION('3', 1), RELATION('4', 1), RELATION('5', 1),

RELATION('6', 1), RELATION('7', 1), RELATION('8', 1),

RELATION('9', 1)),

NODE()

);

FST fstId(LEX\_ID, SYMB\_TABLE\_ID, "", 2,

NODE(26, RELATION('a', 1), RELATION('b', 1), RELATION('c', 1), RELATION('d', 1),

RELATION('e', 1), RELATION('f', 1), RELATION('g', 1), RELATION('h', 1),

RELATION('i', 1), RELATION('j', 1), RELATION('k', 1), RELATION('l', 1),

RELATION('m', 1), RELATION('n', 1), RELATION('o', 1), RELATION('p', 1),

RELATION('q', 1), RELATION('r', 1), RELATION('s', 1), RELATION('t', 1),

RELATION('u', 1), RELATION('v', 1), RELATION('w', 1), RELATION('x', 1),

RELATION('y', 1), RELATION('z', 1)),

NODE()

);

FST fstInt(LEX\_INT, SYMB\_LEXEMS, "", 4,

NODE(1, RELATION('i', 1)),

NODE(1, RELATION('n', 2)),

NODE(1, RELATION('t', 3)),

NODE()

);

FST fstStr(LEX\_STR, SYMB\_LEXEMS, "", 7,

NODE(1, RELATION('s', 1)),

NODE(1, RELATION('t', 2)),

NODE(1, RELATION('r', 3)),

NODE(1, RELATION('i', 4)),

NODE(1, RELATION('n', 5)),

NODE(1, RELATION('g', 6)),

NODE()

);

FST fstFunc(LEX\_FUNC, SYMB\_LEXEMS, "", 9,

NODE(1, RELATION('f', 1)),

NODE(1, RELATION('u', 2)),

NODE(1, RELATION('n', 3)),

NODE(1, RELATION('c', 4)),

NODE(1, RELATION('t', 5)),

NODE(1, RELATION('i', 6)),

NODE(1, RELATION('o', 7)),

NODE(1, RELATION('n', 8)),

NODE()

);

FST fstIF(LEX\_IF, SYMB\_LEXEMS, "", 3,

NODE(1, RELATION('i', 1)),

NODE(1, RELATION('f', 2)),

NODE()

);

FST fstThen(LEX\_THEN, SYMB\_LEXEMS, "", 5,

NODE(1, RELATION('t', 1)),

NODE(1, RELATION('h', 2)),

NODE(1, RELATION('e', 3)),

NODE(1, RELATION('n', 4)),

NODE()

);

FST fstElse(LEX\_ELSE, SYMB\_LEXEMS, "", 5,

NODE(1, RELATION('e', 1)),

NODE(1, RELATION('l', 2)),

NODE(1, RELATION('s', 3)),

NODE(1, RELATION('e', 4)),

NODE()

);

FST fstNew(LEX\_NEW, SYMB\_LEXEMS, "", 4,

NODE(1, RELATION('n', 1)),

NODE(1, RELATION('e', 2)),

NODE(1, RELATION('w', 3)),

NODE()

);

FST fstMain(LEX\_MAIN, SYMB\_LEXEMS, "", 5,

NODE(1, RELATION('m', 1)),

NODE(1, RELATION('a', 2)),

NODE(1, RELATION('i', 3)),

NODE(1, RELATION('n', 4)),

NODE()

);

FST fstOut(LEX\_PRINT, SYMB\_LEXEMS, "", 6,

NODE(1, RELATION('p', 1)),

NODE(1, RELATION('r', 2)),

NODE(1, RELATION('i', 3)),

NODE(1, RELATION('n', 4)),

NODE(1, RELATION('t', 5)),

NODE()

);

FST fstRet(LEX\_RET, SYMB\_LEXEMS, "", 4,

NODE(1, RELATION('r', 1)),

NODE(1, RELATION('e', 2)),

NODE(1, RELATION('t', 3)),

NODE()

);

FST fstIntLit(LEX\_INT, SYMB\_LITERAL, "", 2,

NODE(20, RELATION('0', 0), RELATION('1', 0), RELATION('2', 0),

RELATION('3', 0), RELATION('4', 0), RELATION('5', 0),

RELATION('6', 0), RELATION('7', 0), RELATION('8', 0),

RELATION('9', 0),

RELATION('0', 1), RELATION('1', 1), RELATION('2', 1),

RELATION('3', 1), RELATION('4', 1), RELATION('5', 1),

RELATION('6', 1), RELATION('7', 1), RELATION('8', 1),

RELATION('9', 1)),

NODE()

);

FST fstId(LEX\_ID, SYMB\_TABLE\_ID, "", 2,

NODE(52, RELATION('a', 0), RELATION('b', 0), RELATION('c', 0), RELATION('d', 0),

RELATION('e', 0), RELATION('f', 0), RELATION('g', 0), RELATION('h', 0),

RELATION('i', 0), RELATION('j', 0), RELATION('k', 0), RELATION('l', 0),

RELATION('m', 0), RELATION('n', 0), RELATION('o', 0), RELATION('p', 0),

RELATION('q', 0), RELATION('r', 0), RELATION('s', 0), RELATION('t', 0),

RELATION('u', 0), RELATION('v', 0), RELATION('w', 0), RELATION('x', 0),

RELATION('y', 0), RELATION('z', 0),

RELATION('a', 1), RELATION('b', 1), RELATION('c', 1), RELATION('d', 1),

RELATION('e', 1), RELATION('f', 1), RELATION('g', 1), RELATION('h', 1),

RELATION('i', 1), RELATION('j', 1), RELATION('k', 1), RELATION('l', 1),

RELATION('m', 1), RELATION('n', 1), RELATION('o', 1), RELATION('p', 1),

RELATION('q', 1), RELATION('r', 1), RELATION('s', 1), RELATION('t', 1),

RELATION('u', 1), RELATION('v', 1), RELATION('w', 1), RELATION('x', 1),

RELATION('y', 1), RELATION('z', 1)),

NODE()

);

FST fstStrLit(LEX\_STR, SYMB\_LITERAL, "", 4,

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

NODE(276, \

RELATION('A', 1), RELATION('B', 1), RELATION('C', 1), RELATION('D', 1), RELATION('E', 1), RELATION('F', 1), \

RELATION('G', 1), RELATION('H', 1), RELATION('I', 1), RELATION('J', 1), RELATION('K', 1), RELATION('L', 1), \

RELATION('M', 1), RELATION('N', 1), RELATION('O', 1), RELATION('P', 1), RELATION('Q', 1), RELATION('R', 1), \

RELATION('S', 1), RELATION('T', 1), RELATION('U', 1), RELATION('V', 1), RELATION('W', 1), RELATION('X', 1), \

RELATION('Y', 1), RELATION('Z', 1), \

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

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

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

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

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

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

\

RELATION('А', 1), RELATION('Б', 1), RELATION('В', 1), RELATION('Г', 1), RELATION('Д', 1), RELATION('Е', 1), \

RELATION('Ё', 1), RELATION('Ж', 1), RELATION('З', 1), RELATION('И', 1), RELATION('Й', 1), RELATION('К', 1), \

RELATION('Л', 1), RELATION('М', 1), RELATION('Н', 1), RELATION('О', 1), RELATION('П', 1), RELATION('Р', 1), \

RELATION('С', 1), RELATION('Т', 1), RELATION('У', 1), RELATION('Ф', 1), RELATION('Х', 1), RELATION('Ц', 1), \

RELATION('Ч', 1), RELATION('Ш', 1), RELATION('Щ', 1), RELATION('Ъ', 1), RELATION('Ы', 1), RELATION('Ь', 1), \

RELATION('Э', 1), RELATION('Ю', 1), RELATION('Я', 1), \

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

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

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

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

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

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

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

RELATION('(', 1), \

\

RELATION('A', 2), RELATION('B', 2), RELATION('C', 2), RELATION('D', 2), RELATION('E', 2), RELATION('F', 2), \

RELATION('G', 2), RELATION('H', 2), RELATION('I', 2), RELATION('J', 2), RELATION('K', 2), RELATION('L', 2), \

RELATION('M', 2), RELATION('N', 2), RELATION('O', 2), RELATION('P', 2), RELATION('Q', 2), RELATION('R', 2), \

RELATION('S', 2), RELATION('T', 2), RELATION('U', 2), RELATION('V', 2), RELATION('W', 2), RELATION('X', 2), \

RELATION('Y', 2), RELATION('Z', 2), \

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

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

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

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

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

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

\

RELATION('А', 2), RELATION('Б', 2), RELATION('В', 2), RELATION('Г', 2), RELATION('Д', 2), RELATION('Е', 2), \

RELATION('Ё', 2), RELATION('Ж', 2), RELATION('З', 2), RELATION('И', 2), RELATION('Й', 2), RELATION('К', 2), \

RELATION('Л', 2), RELATION('М', 2), RELATION('Н', 2), RELATION('О', 2), RELATION('П', 2), RELATION('Р', 2), \

RELATION('С', 2), RELATION('Т', 2), RELATION('У', 2), RELATION('Ф', 2), RELATION('Х', 2), RELATION('Ц', 2), \

RELATION('Ч', 2), RELATION('Ш', 2), RELATION('Щ', 2), RELATION('Ъ', 2), RELATION('Ы', 2), RELATION('Ь', 2), \

RELATION('Э', 2), RELATION('Ю', 2), RELATION('Я', 2), \

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

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

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

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

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

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

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

RELATION('(', 2)), \

\

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

NODE()

);

FST fstPow(LEX\_POW, SYMB\_STATIC\_LIB, "", 5,

NODE(1, RELATION('p', 1)), \

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

NODE(1, RELATION('w', 3)), \

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

NODE()

);

FST fstLen(LEX\_LENGTH, SYMB\_STATIC\_LIB, "", 4,

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

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

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

NODE()

);

Листинг 4 – Структура таблицы лексем

struct Entry

{

char lexema; // лексема

int lineNo; // номер строки в исходном коде

int indexTI; // индекс в таблице идентификаторов

short priority; // приоритет для операций

};

struct LexTable

{

int maxize; // емкость таблицы лексем

int size; // текущий размер таблицы лексем

Entry\* table; // массив строк табилцы лексем

};

Листинг 5 – Структура таблицы идентификаторов

namespace IT

{

enum IDDATATYPE { IDDATATYPE\_OFF = 0, IDDATATYPE\_INT = 1, IDDATATYPE\_STR = 2, IDDATATYPE\_MATH = 3}; //OFF - ничего, выбор (начало)

enum IDTYPE { IDTYPE\_N = 0, IDTYPE\_TAB = 1, IDTYPE\_FUN = 2, IDTYPE\_PAR = 3, IDTYPE\_LIT = 4, IDTYPE\_ACT = 5 }; //переменная, функция, параметр функции, литерал, оператор

//таблица идентификаторов

struct Entry

{

int idFirstInLT; //индекс первого вхождения в таблице лексем

string id; //идентификатор (автоматические усекается до ID\_MAXSIZE)

IDDATATYPE idDataType; //тип данных

IDTYPE idType; //тип идентификатора

struct

{

int numValue; //значение int

struct

{

int lenghtstr; //длина строки

string str; //символы string

} rowValue; //значение string

} value; //значение идентификатора

};

struct IdTable //экземпляр таблицы идентификаторов

{

int maxsize; //емкость таблицы идентификаторов < TI\_MAXSIZE

int size; //текущий размер таблицы идентификаторов < maxsize

Entry\* table; //массив строк таблицы идентификаторов

};

IdTable Create(int size); //создать таблицу идентификаторов

void Add(IdTable& idtable, Entry entry); //добавить строку в таблицу идентификаторов

Entry GetEntry(IdTable& idtable, int n); //получить строку таблицы идентификаторов

int IsId(IdTable& idtable, string id); //возврат: номер строки(если есть), TI\_NULLIDX(если нет)

int CheckId(IdTable& idtable, string value);

int IsDublId(IdTable& idtable, string id); //проверка на дублирование ид

int CheckId(IdTable& idtable, int value);

void ShowTable(IdTable& idtable);

void WriteTable(Log::LOG log, IdTable& idtable);

};

# **Приложение Б**

Листинг 1 – Грамматика языка ADS-2018

Greibach greibach(

NS('S'), TS('$'), // стартовый символ, дно стека NS-нетерминал(большие буквы),TS-терминал

8, // количество правил

Rule(

NS('S'), GRB\_ERROR\_SERIES + 0, // неверная структура программы

2, // S->tfi(F){N}S | m{N};

Rule::Chain(10, TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('}'), NS('S')),

Rule::Chain(4, TS('m'), TS('{'), NS('N'), TS('}'))

),

Rule(

NS('N'), GRB\_ERROR\_SERIES + 1, // ошибочный оператор

12, // N->nti;N | N->nti=E;N| rE; | i=E;N | o(i);N | o(s);N |o(x);N |o(s); |o(i);|o(x); | ?Q1{N}2{N}N | ?Q1{N}N

Rule::Chain(5, TS('n'), TS('t'), TS('i'), TS(';'), NS('N')),

Rule::Chain(7, TS('n'), TS('t'), TS('i'), TS('='), NS('E'), TS(';'), NS('N')),

Rule::Chain(3, TS('r'), NS('E'), TS(';')),

Rule::Chain(5, TS('i'), TS('='), NS('E'), TS(';'), NS('N')),

Rule::Chain(6, TS('o'), TS('('), TS('i'), TS(')'), TS(';'), NS('N')),

Rule::Chain(6, TS('o'), TS('('), TS('s'), TS(')'), TS(';'), NS('N')),

Rule::Chain(6, TS('o'), TS('('), TS('x'), TS(')'), TS(';'), NS('N')),

Rule::Chain(5, TS('o'), TS('('), TS('i'), TS(')'), TS(';')),

Rule::Chain(5, TS('o'), TS('('), TS('s'), TS(')'), TS(';')),

Rule::Chain(5, TS('o'), TS('('), TS('x'), TS(')'), TS(';')),

Rule::Chain(11, TS('?'), NS('Q'), TS('1'), TS('{'), NS('N'), TS('}'), TS('2'), TS('{'), NS('N'), TS('}'), NS('N')),

Rule::Chain(7, TS('?'), NS('Q'), TS('1'), TS('{'), NS('N'), TS('}'), NS('N'))

),

Rule(

NS('E'), GRB\_ERROR\_SERIES + 2, // ошибка в выражении

16, // E->i | s | x | (E) | xM | i(W) | iM | nM | (E)M | i(W)M | p(i,x) | p(x,i)| p(x,x) |p(i,i) |c(s) |c(i)

Rule::Chain(1, TS('i')),

Rule::Chain(1, TS('s')),

Rule::Chain(1, TS('x')),

Rule::Chain(3, TS('('), NS('E'), TS(')')),

Rule::Chain(2, TS('x'), NS('M')),

Rule::Chain(4, TS('i'), TS('('), NS('W'), TS(')')),

Rule::Chain(2, TS('i'), NS('M')),

Rule::Chain(2, TS('n'), NS('M')),

Rule::Chain(4, TS('('), NS('E'), TS(')'), NS('M')),

Rule::Chain(5, TS('i'), TS('('), NS('W'), TS(')'), NS('M')),

Rule::Chain(6, TS('p'), TS('('), TS('i'), TS(','), TS('x'), TS(')')),

Rule::Chain(6, TS('p'), TS('('), TS('x'), TS(','), TS('x'), TS(')')),

Rule::Chain(6, TS('p'), TS('('), TS('i'), TS(','), TS('i'), TS(')')),

Rule::Chain(6, TS('p'), TS('('), TS('i'), TS(','), TS('x'), TS(')')),

Rule::Chain(4, TS('c'), TS('('), TS('i'), TS(')')),

Rule::Chain(4, TS('c'), TS('('), TS('s'), TS(')'))

),

Rule(

NS('F'), GRB\_ERROR\_SERIES + 3, // ошибка в параметрах функции

2, // F -> ti | ti,F

Rule::Chain(2, TS('t'), TS('i')),

Rule::Chain(4, TS('t'), TS('i'), TS(','), NS('F'))

),

Rule(

NS('W'), GRB\_ERROR\_SERIES + 4, // ошибка в параметрах вызываемой функции

6, // W -> i | s | x | i,W | s,W | x,W

Rule::Chain(1, TS('i')),

Rule::Chain(1, TS('s')),

Rule::Chain(1, TS('x')),

Rule::Chain(3, TS('i'), TS(','), NS('W')),

Rule::Chain(3, TS('s'), TS(','), NS('W')),

Rule::Chain(3, TS('x'), TS(','), NS('W'))

),

Rule(

NS('M'), GRB\_ERROR\_SERIES + 5, // оператор

3, // M -> a | aE | aEM

Rule::Chain(1, TS('a')),

Rule::Chain(2, TS('a'), NS('E')),

Rule::Chain(3, TS('a'), NS('E'), NS('M'))

),

Rule(

NS('Q'), GRB\_ERROR\_SERIES + 6,

2, // Q-> (R<R) | (R>R)

Rule::Chain(5, TS('('), NS('R'), TS('<'), NS('R'), TS(')')),

Rule::Chain(5, TS('('), NS('R'), TS('>'), NS('R'), TS(')'))

),

Rule(

NS('R'), GRB\_ERROR\_SERIES + 7,

2, // R-> i | x

Rule::Chain(1, TS('i')),

Rule::Chain(1, TS('d'))

)

);

# **Приложение В**

Листинг 1 – Структура конечного автомата

struct MfstState // состояние автомата (для сохранения)

{

short lenta\_position; // позиция на ленте

short ruleNo; // номер текущего правила

short rulechainNo; // номер текущей цепочки текущего правила

MFSTSTSTACK stack; // стек автомата

MfstState();

MfstState(short pposition, MFSTSTSTACK pst, short pnrulechain);

MfstState(

short pposition, // позиция на ленте

MFSTSTSTACK pst, // стек автомата

short pnrule, // номер текущего правила

short pnrulechain // номер текущей цепочки

);

};

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 (ошибка в грамматике)

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

NS\_ERROR, // неизвестный нетерминальный символ грамматики

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

TS\_NOK, // тек. 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 ruleNo; // номер правила

short rule\_chainNo; // 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 ruleNo; // номер текущего правила

short rulechainNo; // номер текущей цепочки текущего правила

short lenta\_size; // размер ленты

GRB::Greibach greibach; // грамматика Грейбах

Lex::LEX lex; // результат работы лексического анализатора

MFSTSTSTACK stack; // стек автомата

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

Mfst();

Mfst(

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

GRB::Greibach pgrebach // грамматика Грейбах

);

char\* getContainStack(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 строку диагностики или 0х00

bool saveState(Log::LOG log); // сохранить состояние автомата

bool restState(Log::LOG log); // восстановить состояние автомата

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

GRB::Rule::Chain chain // цепочка правил

);

Mfst::RC\_STEP step(Log::LOG log);// выполнить шаг автомата

bool start(Log::LOG log); // запустить автомат

bool saveDiagnosis(

RC\_STEP pprc\_step // код завершения шага

);

void printRules(Log::LOG &log); // вывести последовательность правил (дерево разбора)

};

# **Приложение Г**

Листинг 1 – Разбор синтаксическим анализатором исходного кода

Начало разбора

Шаг :Правило Входная лента Стек

0 : S->tfi(F){N}S tfi(ti,ti){nti;i=(iai)ax; S$

0 : SAVESTATE: 1

0 : tfi(ti,ti){nti;i=(iai)ax; tfi(F){N}S$

1 : fi(ti,ti){nti;i=(iai)ax;r fi(F){N}S$

2 : i(ti,ti){nti;i=(iai)ax;ri i(F){N}S$

3 : (ti,ti){nti;i=(iai)ax;ri; (F){N}S$

4 : ti,ti){nti;i=(iai)ax;ri;} F){N}S$

5 : F->ti ti,ti){nti;i=(iai)ax;ri;} F){N}S$

5 : SAVESTATE: 2

5 : ti,ti){nti;i=(iai)ax;ri;} ti){N}S$

6 : i,ti){nti;i=(iai)ax;ri;}t i){N}S$

7 : ,ti){nti;i=(iai)ax;ri;}tf ){N}S$

8 : TS\_NOK/NS\_NORULECHAIN

8 : RESTATE

8 : ti,ti){nti;i=(iai)ax;ri;} F){N}S$

9 : F->ti,F ti,ti){nti;i=(iai)ax;ri;} F){N}S$

9 : SAVESTATE: 2

9 : ti,ti){nti;i=(iai)ax;ri;} ti,F){N}S$

10 : i,ti){nti;i=(iai)ax;ri;}t i,F){N}S$

11 : ,ti){nti;i=(iai)ax;ri;}tf ,F){N}S$

12 : ti){nti;i=(iai)ax;ri;}tfi F){N}S$

13 : F->ti ti){nti;i=(iai)ax;ri;}tfi F){N}S$

13 : SAVESTATE: 3

Конец разбора

412 : SAVESTATE: 64

412 : o(s);rx;} o(s);N}$

413 : (s);rx;} (s);N}$

414 : s);rx;} s);N}$

415 : );rx;} );N}$

416 : ;rx;} ;N}$

417 : rx;} N}$

418 : N->rE; rx;} N}$

418 : SAVESTATE: 65

418 : rx;} rE;}$

419 : x;} E;}$

420 : E->x x;} E;}$

420 : SAVESTATE: 66

420 : x;} x;}$

421 : ;} ;}$

422 : } }$

423 : $

424 : --------------LENTA\_END

425 : ----------------------------->LENTA\_END

Листинг 2 - Правила разбора

0 : S->tfi(F){N}S

4 : F->ti,F

7 : F->ti

11 : N->nti;N

15 : N->i=E;N

17 : E->(E)M

18 : E->iM

19 : M->aE

20 : E->i

22 : M->aE

23 : E->x

25 : N->rE;

26 : E->i

29 : S->tfi(F){N}S

33 : F->ti

37 : N->nti;N

41 : N->i=E;N

43 : E->c(i)

48 : N->rE;

49 : E->i

52 : S->tfi(F){N}S

56 : F->ti,F

59 : F->ti

63 : N->nti;N

67 : N->i=E;N

69 : E->p(i,i)

76 : N->rE;

77 : E->i

80 : S->m{N}

82 : N->nti;N

86 : N->nti=E;N

90 : E->x

92 : N->nti;N

96 : N->nti;N

100 : N->i=E;N

102 : E->(E)M

103 : E->xM

104 : M->aE

105 : E->x

107 : M->aE

108 : E->x

110 : N->i=E;N

112 : E->i(W)

114 : W->i,W

116 : W->i

119 : N->?Q1{N}2{N}N

120 : Q->(R<R)

121 : R->i

123 : R->i

127 : N->o(i);

135 : N->o(i);

141 : N->o(i);N

146 : N->i=E;N

148 : E->s

150 : N->i=E;N

152 : E->i(W)

154 : W->i

157 : N->o(i);N

162 : N->i=E;N

164 : E->i(W)

166 : W->x,W

168 : W->x

171 : N->o(i);N

176 : N->o(s);N

181 : N->rE;

182 : E->x

# **Приложение Д**

Листинг 1 – Алгоритм преобразования выражений к польской записи

void CallPolishNotation(LT::LexTable\* lextable, IT::IdTable\* idtable)

{

for (int i = 0; i < lextable->size; i++) {

if (lextable->table[i].lexema == LEX\_EQUAL)

{

PolishNotation(++i, lextable, idtable);

}

}

}

void AddToResult(IT::IdTable\* idtable, LT::Entry\* result, LT::Entry elementLT, int \*pos, int \*flag, int lenout, int lextable\_pos)

{

if (idtable->table[elementLT.indexTI].idFirstInLT >= lextable\_pos && idtable->table[elementLT.indexTI].idFirstInLT < (lextable\_pos + lenout))

{

idtable->table[elementLT.indexTI].idFirstInLT = lextable\_pos + \*flag;

}

\*flag += 1;

result[\*pos] = elementLT;

\*pos += 1;

}

void AddToResult(stack<LT::Entry>\* stk, IT::IdTable\* idtable, LT::Entry\* result, int \*pos, int \*flag, int lenout, int lextable\_pos)

{

if (idtable->table[stk->top().indexTI].idFirstInLT >= lextable\_pos && idtable->table[stk->top().indexTI].idFirstInLT < (lextable\_pos + lenout))

{

idtable->table[stk->top().indexTI].idFirstInLT = lextable\_pos + \*flag;

}

\*flag += 1;

result[\*pos] = stk->top();

\*pos += 1;

stk->pop();

}

bool PolishNotation(int lextable\_pos, LT::LexTable\* lextable, IT::IdTable\* idtable)

{

stack<LT::Entry> stackLTelements;

LT::Entry\* elementsLT = new LT::Entry[lextable->size];

int ncomma = 0,

flag = 0,

waste = 0,

funcPositionTI = -1,

lenght = 0, //общая длина

lenout = 0, //длина выходной строки

semicolonid; //ид для элемента таблицы с точкой с запятой

LT::Entry tempLTEntry, bufEntry;

for (int i = lextable\_pos; lextable->table[i].lexema != LEX\_SEMICOLON; i++) {

lenout = i + 1;

semicolonid = i + 1;

}

for (int i = lextable\_pos; i < lenout; i++) {

tempLTEntry = lextable->table[i];

if (tempLTEntry.lexema == LEX\_ID || tempLTEntry.lexema == LEX\_INT || tempLTEntry.lexema == LEX\_STR ||

tempLTEntry.lexema == LEX\_POW || tempLTEntry.lexema == LEX\_LENGTH)

{

if (idtable->table[tempLTEntry.indexTI].idType == IT::IDTYPE\_FUN)

{

funcPositionTI = tempLTEntry.indexTI;

continue;

}

AddToResult(idtable, elementsLT, tempLTEntry, &lenght, &flag, lenout, lextable\_pos);

}

else {

if (tempLTEntry.lexema == LEX\_ACTION)

{

while (!stackLTelements.empty() && stackLTelements.top().priority >= tempLTEntry.priority)

{

if (stackLTelements.top().lexema != LEX\_LEFTHESIS)

{

AddToResult(&stackLTelements, idtable, elementsLT, &lenght, &flag, lenout, lextable\_pos);

}

else break;

}

stackLTelements.push(tempLTEntry);

}

if (tempLTEntry.lexema == LEX\_COMMA)

{

ncomma++;

while (stackLTelements.top().lexema == LEX\_ACTION) {

AddToResult(&stackLTelements, idtable, elementsLT, &lenght, &flag, lenout, lextable\_pos);

}

}

else if (tempLTEntry.lexema != LEX\_RIGHTHESIS)

{

if (stackLTelements.empty() || stackLTelements.top().lexema == LEX\_LEFTHESIS || tempLTEntry.lexema == LEX\_LEFTHESIS)

{

stackLTelements.push(tempLTEntry);

}

}

if (tempLTEntry.lexema == LEX\_RIGHTHESIS && tempLTEntry.priority != 4)

{

waste += 2;

while (stackLTelements.top().lexema != LEX\_LEFTHESIS) {

AddToResult(&stackLTelements, idtable, elementsLT, &lenght, &flag, lenout, lextable\_pos);

}

stackLTelements.pop();

}

if (tempLTEntry.lexema == LEX\_RIGHTHESIS && tempLTEntry.priority == 4)

{

bufEntry.lexema = LEX\_SUBST;

bufEntry.indexTI = funcPositionTI;

bufEntry.lineNo = elementsLT[lenght - 1].lineNo;

bufEntry.priority = ncomma + 1; //здесь хранится кол-во параметров в функции

elementsLT[lenght++] = bufEntry;

if (ncomma != 0)

{

waste += ncomma;

ncomma = 0;

}

while (stackLTelements.top().lexema != LEX\_LEFTHESIS) {

AddToResult(&stackLTelements, idtable, elementsLT, &lenght, &flag, lenout, lextable\_pos);

}

stackLTelements.pop();

waste += 2;

}

}

}

while (!stackLTelements.empty()) {

AddToResult(&stackLTelements, idtable, elementsLT, &lenght, &flag, lenout, lextable\_pos);

}

for (int i = lextable\_pos, k = 0; i < lextable\_pos + lenght; i++, k++) {

lextable->table[i] = elementsLT[k]; //запись в таблицу польской записи

}

lextable->table[lextable\_pos + lenght] = lextable->table[semicolonid]; //вставка элемента с точкой с запятой

for (int i = 0; i < waste; i++) {

lextable->size--;

for (int j = lextable\_pos + lenght + 1; j < lextable->size; j++) { // сдвигаем на удалённые литералы

lextable->table[j] = lextable->table[j + 1];

if (lextable->table[j].indexTI != TL\_TI\_NULLIDX && idtable->table[lextable->table[j].indexTI].idFirstInLT == (j + 1))

{

idtable->table[lextable->table[j].indexTI].idFirstInLT -= 1;

}

}

}

return true;

}

Листинг 2 – Преобразованная в результате польской записи таблица лексем

1: tfi(ti,ti){

2: nti;

3: i=iiaxa;

4: ri;

5: }

6: tfi(ti){

7: nti;

8: i=i@;

9: ri;

10: }

11: tfi(ti,ti){

12: nti;

13: i=ii@;

14: ri;

15: }

16: m{

17: nti;

18: nti=x;

19: nti;

20: nti;

21: i=xxaxa;

22: i=ii@;

23: ?(i<i)1{

24: o(i);

25: }

26: 2{

27: o(i);

28: }

29: o(i);

30: i=s;

31: i=i@;

32: o(i);

33: i=xx@;

34: o(i);

35: o(s);

36: rx;

37: }

# **Приложение Е**

Листинг 1 – Генерация кода на язык MASM

.586

.model flat, stdcall

includelib libucrt.lib

includelib kernel32.lib

includelib ../Debug/Library.lib

ExitProcess PROTO :DWORD

printi PROTO: DWORD

prints PROTO: DWORD

len PROTO: DWORD

pows PROTO: DWORD, :DWORD

.stack 4096

.const

L0 DWORD 2

L1 DWORD 59

L2 DWORD 17

L3 DWORD 9

L4 BYTE "My name is Diana", 0

L5 DWORD 6

L6 BYTE "Hello World!!!", 0

L7 DWORD 0

.data

onec DWORD ?

twoe DWORD ?

threez DWORD ?

maine DWORD ?

mainf DWORD ?

maing DWORD ?

mainresult DWORD ?

.code

one PROC onea : SDWORD, oneb : SDWORD

push onea

push oneb

pop eax

pop ebx

add eax, ebx

push eax

push L0

pop eax

pop ebx

mul ebx

push eax

pop onec

push onec

jmp local0

local0:

pop eax

ret

one ENDP

two PROC twod : DWORD

push twod

pop edx

push twod

call len

push eax

pop twoe

push twoe

jmp local1

local1:

pop eax

ret

two ENDP

three PROC threex : SDWORD, threey : SDWORD

push threex

push threey

pop edx

pop edx

push threey

push threex

call pows

push eax

pop threez

push threez

jmp local2

local2:

pop eax

ret

three ENDP

main PROC

push L1

pop mainf

push L0

push L2

pop eax

pop ebx

mul ebx

push eax

push L3

pop eax

pop ebx

add eax, ebx

push eax

pop maine

push maine

push mainf

pop edx

pop edx

push mainf

push maine

call one

push eax

pop maing

mov eax, maine

cmp eax, maing

jl m0

jg m1

je m1

m0:

push maine

call printi

jmp e0

m1:

push maing

call printi

e0:

push maing

call printi

push offset L4

pop mainresult

push mainresult

pop edx

push mainresult

call two

push eax

pop maing

push maing

call printi

push L3

push L5

pop edx

pop edx

push L5

push L3

call three

push eax

pop maing

push maing

call printi

push offset L6

call prints

push 0

jmp theend

theend:

call ExitProcess

main ENDP

end main

Листинг 2 – Результат работы транслятора

