Оглавление

[Введение 5](#_Toc122623217)

[1.Спецификация языка программирования 6](#_Toc122623218)

[1.1 Характеристика языка программирования 6](#_Toc122623219)

[1.2 Алфавит языка 6](#_Toc122623220)

[1.3 Символы-сепараторы 6](#_Toc122623221)

[1.4 Применяемые кодировки 7](#_Toc122623222)

[1.5 Типы данных 7](#_Toc122623223)

[1.6 Преобразование типов данных 8](#_Toc122623224)

[1.7 Идентификаторы 8](#_Toc122623225)

[1.8 Литералы 8](#_Toc122623226)

[1.9 Область видимости идентификаторов 9](#_Toc122623227)

[1.10 Инициализация данных 9](#_Toc122623228)

[1.11 Инструкции языка 9](#_Toc122623229)

[1.12 Операции языка 10](#_Toc122623230)

[1.13 Выражения и их вычисления 11](#_Toc122623231)

[1.14 Программные конструкции языка 11](#_Toc122623232)

[1.15 Область видимости 11](#_Toc122623233)

[1.16 Семантические проверки 11](#_Toc122623234)

[1.17 Распределение оперативной памяти на этапе выполнения 12](#_Toc122623235)

[1.18 Стандартная библиотека и её состав 12](#_Toc122623236)

[1.19 Ввод и вывод данных 12](#_Toc122623237)

[1.20 Точка входа 13](#_Toc122623238)

[1.21 Препроцессор 13](#_Toc122623239)

[1.22 Соглашение о вызовах 13](#_Toc122623240)

[1.23 Объектный код 13](#_Toc122623241)

[1.24 Классификация сообщений транслятора 13](#_Toc122623242)

[1.25 Контрольный пример 13](#_Toc122623243)

[2.Структура транслятора 14](#_Toc122623244)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 14](#_Toc122623245)

[2.2 Перечень входных параметров транслятора 14](#_Toc122623246)

[2.3 Перечень протоколов, формируемых транслятором и из содержимое 15](#_Toc122623247)

[3.Разработка лексического анализатора 16](#_Toc122623248)

[3.1 Структура лексического анализатора 16](#_Toc122623249)

[3.2 Контроль входных символов 16](#_Toc122623250)

[3.3 Удаление избыточных символов 17](#_Toc122623251)

[3.4 Перечень ключевых слов 17](#_Toc122623252)

[3.5 Основные структуры данных 18](#_Toc122623253)

[3.6 Структура и перечень сообщений лексического анализатора 18](#_Toc122623254)

[3.7 Принцип обработки ошибок 19](#_Toc122623255)

[3.8 Параметры лексического анализатора 19](#_Toc122623256)

[3.9 Алгоритм лексического анализа 19](#_Toc122623257)

[3.10 Контрольный пример 19](#_Toc122623258)

[4.Разработка синтаксического анализатора 20](#_Toc122623259)

[4.1 Структура синтаксического анализатора 20](#_Toc122623260)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис 20](#_Toc122623261)

[4.3 Построение конечного магазинного автомата 21](#_Toc122623262)

[4.4 Основные структуры данных 22](#_Toc122623263)

[4.5 Описание алгоритма синтаксического разбора 22](#_Toc122623264)

[4.6 Структура и перечень сообщений синтаксического анализатора 22](#_Toc122623265)

[4.7 Параметры синтаксического анализатора и режимы его работы 23](#_Toc122623266)

[4.8 Принцип обработки ошибок 23](#_Toc122623267)

[4.9 Контрольный пример 23](#_Toc122623268)

[**5. Разработка семантического анализатора** 24](#_Toc122623269)

[5.1 Структура семантического анализатора 24](#_Toc122623270)

[5.2 Функции семантического анализатора 24](#_Toc122623271)

[5.3 Перечень сообщений семантического анализатора 24](#_Toc122623272)

[5.4 Принцип обработки ошибок 25](#_Toc122623273)

[5.5 Контрольный пример 25](#_Toc122623274)

[6. Преобразование выражений 26](#_Toc122623275)

[6.1 Выражения, допускаемые языком 26](#_Toc122623276)

[6.2 Польская запись 26](#_Toc122623277)

[7. Генерация кода 27](#_Toc122623278)

[7.1 Структура генератора кода 27](#_Toc122623279)

[7.2 Представление типов данных в оперативной памяти 27](#_Toc122623280)

[7.3 Алгоритм работы генератора кода 27](#_Toc122623281)

[8. Тестирование транслятора 29](#_Toc122623282)

[8.1 Тестирование контрольного примера 29](#_Toc122623283)

[8.2 Тестирование программ с ошибками 29](#_Toc122623284)

[Заключение 31](#_Toc122623285)

[Список использованных источников 32](#_Toc122623286)

[Приложения 33](#_Toc122623287)

[Контрольный пример 33](#_Toc122623288)

[Приложение А 33](#_Toc122623289)

[Приложение Б 42](#_Toc122623290)

[Приложение В 43](#_Toc122623291)

[Приложение Г 44](#_Toc122623292)

[Приложение Д 49](#_Toc122623293)

# Введение

Целью курсовой работы является разработка транслятора BKV-2022.

Для достижения цели сформулированы следующие задачи.

1. Создание лексического анализатора используя конечные автоматы.
2. Создание синтаксического анализатора используя автомат с магазинной памятью.
3. Создание программы по преобразованию выражения в обратную польскую нотацию .
4. Создание генератора кода, генерирующего объектный код на языке Assembler
5. Тестирование транслятора.

Решения каждой из поставленных задач буду приведены в соответствующих главах курсового проекта, а именно :

3) лексический анализатор;

4) синтаксический анализатор;

5) семантический анализатор;

6) преобразование выражений;

7) генерация кода;

8) тестирование транслятора.

Язык программирования BKV-2022 предназначен для работы с консолью, выполнения простейших арифметический действий, операций над строками и логическими переменными.

# 1.Спецификация языка программирования

## 1.1 Характеристика языка программирования

Язык BKV-2022 – компилируемый, строго типизированный язык программирования высокого уровня со статической типизацией, который поддерживает следующие парадигмы программирования: процедурное, модульное, структурное программирование.

## 1.2 Алфавит языка

Совокупность символов, используемых в языке, называется алфавитом языка.

Исходный код BKV-2022 может содержать символы латинского алфавита, цифры десятичной системы счисления от 0 до 9, символы арифметических, сдвиговых и логических операций, а также символы-разделители русские символы разрешены только в строковых литералах. Алфавит языка BKV-2022 формально представлен как совокупность правил в форме Бэкуса-Наура в таблице 1.1.

Таблица 1.1 – Алфавит языка программирования BKV-2022

|  |
| --- |
| Правило |
| строчная буква латинского алфавита = a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z |
| прописная буква латинского алфавита=A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z |
| цифра= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| строчная буква русского алфавита= а|б|в|г|д|е|ё|ж|з|и|й|к|л|м|н|о|п|р|с|т|у|ф|х|ц|ч|ш|щ|ъ|ь|э|ю|я |
| прописная буква русского алфавита= А|Б|В|Г|Д|Е|Ё|Ж|З|И|Й|К|Л|М|Н|О|П|Р|С|Т|У|Ф|Х|Ц|Ч|Ш|Щ|Ъ|Ь|Э|Ю|Я |
| арифметическая операция= +|-|\*|/ |
| символ- сепаратор= ' '|,|(|)|{|}|;|: |
| логическая операция= >|<|==|!= |

## 1.3 Символы-сепараторы

Символы-сепараторы используются для разделения конструкций исходного кода на языке BKV-2022. Они разделены на подгруппы в соответствии с логикой выполняемых ими функций. Символы-сепараторы представлены в таблице 1.2.

Таблица 1.2 – Сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение |
| ;  ' ' (пробел) | Разделение инструкций |

|  |  |
| --- | --- |
| Сепаратор | Назначение |
| , | Перечисление |
| >  <  ==  != | Символы сравнения |
| = | Оператор присвоения |
| +  -  \*  / | Арифметические операции |
| {  } | Программный блок инструкций |
| (  ) | Параметры функций / приоритетность операций (в выражениях) |

## 1.4 Применяемые кодировки

Для написания кода на языке программирования BKV-2022 используется кодировка Windows-1251.

## 1.5 Типы данных

В языке BKV-2022 поддерживается 4 примитивных типов данных: целочисленный, строковый, булевый, символьный. Описание типов данных, предусмотренных в данном языке, представлено в таблице 1.3.

Таблица 1.3 – Типы данных языка BKV-2022

|  |  |  |  |
| --- | --- | --- | --- |
| Тип данных | Описание типа данных | Размер в байтах | Диапазон допустимых значений |
| int | Примитивный тип данных. Предусмотрен для объявления целочисленных данных.  Автоматически инициализируется нулевым значением.  Диапазон допустимых значений: | 4 байта | От -231  до 231 - 1 |
| char | Фундаментальный тип данных. Предусмотрен для объявления символов. Автоматически инициализируется символом ‘\0’. | 1 байт | От -127 до 128 |
| str | Фундаментальный тип данных. Предусмотрен для объявления строк. Автоматическая инициализация строкой нулевой длины. | Последовательность в 255 символов, обязательно оканчивающаясянуль-символом '\0' | - |
| bool | Фундаментальный тип данных. Предусмотрен для объявления логической переменной, которая имеет одно из двух значений: true, false. Автоматическая инициализация значением false. | 1 байт | - |

## 1.6 Преобразование типов данных

Язык BKV-2022 не предоставляет возможности примитивного явного преобразования двух типов данных.

## 1.7 Идентификаторы

В имени идентификатора допускаются символы латинского алфавита верхнего и нижнего регистра. Перед именем каждого идентификаторы функции должно стоят ключевое слово «function». Правило для построения идентификатора в форме Бэкуса-Наура представлено формулой:

<идентификатор> ::= <значение> | <прописная буква латинского алфавита> { (<прописная буква латинского алфавита> | <строчная буква латинского алфавита> | <цифра > ) }

## 1.8 Литералы

В языке существует 4 типа литералов: целого, символьного, логического типов, строкового.

Краткое описание литералов представлено в таблице 1.5.

Таблица 1.5 – Описание литералов

|  |  |
| --- | --- |
| Тип литерала | Описание |
| Литералы целого типа | Целочисленные неотрицательные литералы, инициализируются 0. Литералы только rvalue. |
| Символьные литералы | Символы, заключённые в ' '(одинарные кавычки), инициализируются нуль символом, символьные переменные. Только rvalue. |
| Строковые литералы | Символы, заключённые в '' ''(двойные кавычки), инициализируются пустой строкой, строковые переменные. Только rvalue. |
| Логические литералы | Логический литерал, true/false. Только rvalue. |

## 1.9 Область видимости идентификаторов

Область видимости «сверху вниз». В языке BKV-2022 требуется обязательное объявление переменной перед её использованием. Имеется возможность объявления одинаковых переменных в разных блоках и функциях. Каждая переменная получает префикс – название функции или блока, в котором она объявлена.

## 1.10 Инициализация данных

В момент объявления переменной она инициализируется фиксированным значением. Способы инициализации и их описание представлено в таблице 1.6.

Таблица 1.6 – Способы инициализации переменных

|  |
| --- |
| Вид инициализации |
| <тип данных> <идентификатор>; |

## 1.11 Инструкции языка

Возможные инструкции языка программирования BKV-2022 представлены в общем виде в таблице 1.7.

Таблица 1.7 – Инструкции языка программирования BKV-2022

|  |  |
| --- | --- |
| Инструкция | Запись на языке BKV-2022 |
| Объявление переменной | <тип данных> <идентификатор>; |
| Присваивание | <значение> | <прописная буква латинского алфавита> {(<прописная буква латинского алфавита> | <строчная буква латинского алфавита> | <цифра >)}; |
| Объявление внешней функции | <тип данных> function <идентификатор>(<тип данных> <идентификатор>, …) {…}; |
| Блок инструкций | {  …  } |
| Вывод данных | out (<идентификатор> | <литерал> | <выражение>); |
| Оператор условия | if ( <условие>)  {…} |

## 1.12 Операции языка

В языке программирования BKV-2022 можно выполнять операции сравнения, представленные в таблице 1.8.

Таблица 1.8 – Приоритетности операций языка программирования

(1 – максимальная приоритетность, 8 – минимальная приоритетность)

|  |  |  |
| --- | --- | --- |
| Операция | Приоритетность операции | Тип оператора (унарный, бинарный) |
| Арифметические операции | | |
| () (круглые скобки) | 1 | - |
| \* (умножение) | 2 | бинарный |
| + (бинарный плюс)  - (бинарный минус) | 3 | бинарный |
| Операции сравнения | | |
| < (сравнение: меньше чем)  > (сравнение: больше чем) | 5 | бинарный |
| == (сравнение: равно)  != (сравнение: не равно) | 6 | бинарный |
| Другое | | |
| = (присваивание) | 7 | бинарный |
| , (оператор запятая) | 8 | ­- |

## 1.13 Выражения и их вычисления

Круглые скобки используются при записи математических выражений, обязаны использоваться при объявлении функции, либо для составления сложных сравнений в условиях сравнения. Фигурные скобки используются для составления блоков функций, операторов.

## 1.14 Программные конструкции языка

Язык BKV-2022 предполагает наличие программных конструкций, описанных в таблице 1.9.

Таблица 1.9 – Программные конструкции языка

|  |  |
| --- | --- |
| Называние конструкции | Структура конструкции |
| Главная функция (точка входа) | main  {  return <int -идентификатор> или <int -литерал>;  }; |
| Функции | <тип данных> function<идентификатор> (  <тип данных> <идентификатор>, ...)  {  return <идентификатор> или <литерал>;  }; |
| Блок | {  …  } |

## 1.15 Область видимости

Переменные могут находиться внутри программного блока функции. Вызов функций в глобальной области видимости запрещен.

## 1.16 Семантические проверки

Семантические проверки накладывают ограничения на логику написания исходного кода. Таблица с перечнем семантических проверок, предусмотренных языком, приведена в таблице 1.10.

Таблица 1.10 – Семантические проверки

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Имя идентификатора не может быть ключевым словом |
| 2 | Тип данных переменной должен совпадать с типом значения, которое присваивается этому типу |
| 3 | Несоответствие параметров вызываемой функции с её объявлением |
| 4 | Операнды в арифметическом или логическом выражениях не могут быть разных типов |
| 5 | Тип данных возвращаемой функции должен совпадать с типом при ее объявлении |
| 6 | Невозможно использование строк в целочисленных выражениях |
| 7 | Недопустимость некоторых операция для определённых типов |
| 8 | Недопустимое значение переменных |
| 9 | Запрет на присваивание значения идентификатору функции |

## 1.17 Распределение оперативной памяти на этапе выполнения

Все переменные размещаются в стеке. Таблица лексем и таблица идентификаторов сохраняются в структуры с выделенной под них динамической памятью, которая очищается по окончанию работы транслятора.

## 1.18 Стандартная библиотека и её состав

Функции стандартной библиотеки представлены в таблице 1.11. Стандартная библиотека написана на языке программирования С++. Стандартная библиотека должна быть расположена в директории с кодом на ассемблере и имеет имя файла “lib.txt”

Таблица 1.11 – Состав стандартной библиотеки

|  |  |  |  |
| --- | --- | --- | --- |
| Имя функции | Возвращаемое значение | Принимаемые параметры | Описание |
| cmpr | string | string s1 – строка  string s2 – строка | Возвращает лексикографическое сравнение строк |
| strl | int | string s – строка | Возвращает длину строки |

Вывод в языке осуществлён программно. Присутствует возможность использования манипуляторов вывода строк и вывода целочисленных значений в десятичной и шестнадцатеричной системах счисления.

## 1.19 Ввод и вывод данных

Ввод данных не поддерживается.

out (<идентификатор или литерал>); – вывод в стандартный поток вывода.

## 1.20 Точка входа

Точка входа – это место в исходном коде программы, с которого начинается выполнение программы. Точкой входа является функция main.

## 1.21 Препроцессор

Предусмотрено подключение пользовательских библиотек через директиву import < ”…”|<…> >. Если подключаемый файл указан в <>, то поиск будет происходить в стандартном каталоге “Library”, предназначенный для хранения заголовочных файлов. В случае, если подключаемый файл заключен в двойные кавычки, поиск будет происходить в текущем рабочем каталоге.

## 1.22 Соглашение о вызовах

В языке вызов функций происходит по соглашению о вызовах stdcall.

## 1.23 Объектный код

BKV-2022 транслируется в язык ассемблера.

## 1.24 Классификация сообщений транслятора

При появлении ошибки в коде программы на языке BKV-2022 и выявлении её транслятором, в текущий файл протокола выводится сообщение. Классификация ошибок приведена в таблице 1.13.

Таблица 1.13. – Классификация сообщений транслятора

|  |  |
| --- | --- |
| Интервал | Описание ошибок |
| 0 – 100 | Системные ошибки, ошибки параметров |
| 100 – 300 | Ошибки лексического анализа |
| 600 - … | Ошибки синтаксического анализа |
| 100 – 600 | Ошибки семантического анализа |

## 1.25 Контрольный пример

Контрольный пример представлен в главе «Приложения».

# 2.Структура транслятора

## 2.1 Компоненты транслятора, их назначение и принципы взаимодействия

Задача переводчика состоит в том, чтобы преобразовать программу, написанную на языке BKV-2022, в программу на языке ассемблера. Компонентами транслятора являются лексический анализатор, синтаксический анализатор, семантический анализатор и генератор кода на языке ассемблера.

Исходный код поступает на вход лексического анализатора. Конечным результатом работы транслятора является ассемблерный код.

Лексический анализ является первым этапом трансляции. Результатом работы лексического анализатора является таблица токенов и таблица идентификаторов.

Синтаксический анализ – основная часть транслятора, предназначенная для распознавания синтаксической структуры языка. Входным параметром является таблица токенов.

Синтаксический анализатор распознает синтаксические ошибки и создает дерево разбора.

Семантический анализ проверяет семантическую непротиворечивость языковых конструкций исходного кода и проверяет правильность текста исходной программы с семантической точки зрения.

Генератор кода — это этап транслятора, который генерирует ассемблерный код на основе данных, полученных на предыдущих этапах трансляции. Входными параметрами генератора кода являются таблица идентификаторов и таблица токенов, в которых код языка BKV-2022 преобразуется в код языка Ассемблера

## 2.2 Перечень входных параметров транслятора

Входные параметры транслятора представлены в таблице 2.1.

Таблица 2.1 – Входные параметры транслятора языка

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<имя\_файла> | Файл с исходним кодом программы с расширением .txt | Не предусмотрено |
| -log:<имя\_файла> | Будет определять файл, содержащие результат работы программы. | <имя\_in\_файла>.log.txt |
| -out:<имя\_файла> | Файл для записи результата работы транслятора | <имя\_in\_файла>.asm |

## 2.3 Перечень протоколов, формируемых транслятором и из содержимое

Результатом работы транслятора языка BKV-2022 является исходный код на языке ассемблера и протокол работы транслятора, содержащий основную информацию о процессе обработки исходного кода. Протокол работы транслятора содержит исходный код программы, число символов и строк исходного кода, а также информацию о каждой стадии его работы. Таблица контроля входных символов представлена на рисунке 3.1.

Таблица с перечнем протоколов, формируемых транслятором языка BKV-2022 и их назначением представлена в таблице 2.2

Таблица 2.2 – Протоколы, формируемые транслятором языка BKV-2022

|  |  |
| --- | --- |
| Протокол | Описание протокола |
| Файл журнала с параметром <log> | Содержит информацию о входных параметрах приложения, этапе проверки символов на допустимость, результат работы семантического анализатора. |
| Файл вывода лексем <lех> | Содержит результат работы лексического анализатора. |
| Файл вывода литералов <it> | Содержит результат работы синтаксического анализатора. |
| Выходной файл c параметром <out> | Содержит сгенерированный код на языке Ассемблера. |

# 3.Разработка лексического анализатора

## 3.1 Структура лексического анализатора

Лексический анализатор – часть транслятора, выполняющая лексический анализ.

Лексический анализатор преобразует исходный текст, заменяя лексические единицы языка их внутренним представлением – лексемами. Для описания лексики языка программирования применяются регулярные грамматики, относящиеся к типу 3 иерархии Хомского. Язык, заданный регулярной грамматикой, называется регулярным языком (типа 3 иерархии Хомского). Регулярный язык однозначно задается регулярным выражением, а распознавателями для регулярных языков являются конечные автоматы. Грамматики типа 3: GIII = <T,N,P,S> - регулярные грамматики. Регулярные грамматики бывают праволинейными и леволинейными.

Правила праволинейной грамматики:

A или A, где A,B N, \*.

Правила леволинейной грамматики:

A или A, где A,B N, \*.

Входными данными для лексического анализатора является массив символов, предварительно сформированный при считывании исходного файла. Выходными данными являются таблица лексем, таблица идентификаторов, а также протокол работы. Структура лексического анализатора BKV-2022 представлена на рисунке 3.1.

## 3.2 Контроль входных символов

Таблица контроля входных символов представлена на рисунке 3.1

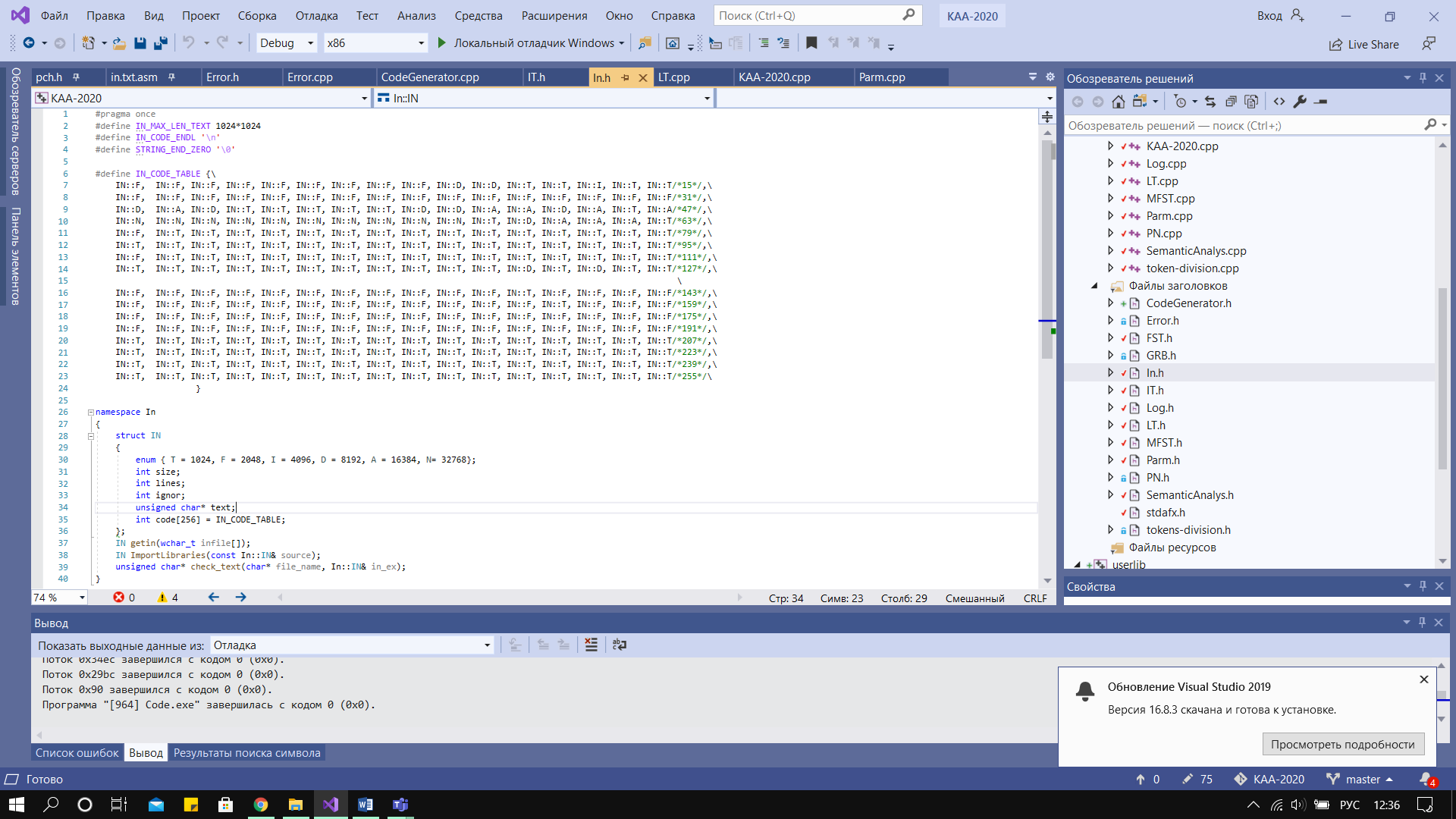


Рисунок 3.1 – Таблица контроля входных символов

Принцип работы таблицы заключается в соответствии значения каждого элемента значению в таблице ASCII.

Описание значения символов: T – разрешённый символ, F – запрещённый символ, I – игнорируемый символ, A – символ арифметической операции, N – символ цифры, D – символ-разделитель.

## 3.3 Удаление избыточных символов

Избыточными символами являются символы табуляции и пробелы.

Избыточные символы удаляются на этапе разбиения исходного кода на лексемы.

## 3.4 Перечень ключевых слов

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 | | int | | t |
| char | | t |
| string | | t |
| bool | | t |
| Ключевые слова | function | | | f |
| out | | | o |
| main | | | i |
| strl | | | i |
| return | | | r |
| Идентификатор | | | i |
| Литерал | | | l |
| Иное | ; | | | ; |
|  | , | | | , |
| Сепараторы | { | | { | |
| } | | } | |
| ( | | ( | |
| ) | | ) | |
| = | | = | |
| - | | v | |
| \* | | v | |
| Операторы | + | | v | |
| / | | v | |
| != | | ! | |
| == | | ~ | |
| > | | > | |
| < | | < | |

Пример реализации таблицы лексем представлен в приложении А.

Также в приложении А находятся конечные автоматы, соответствующие лексемам языка BKV-2022.

## 3.5 Основные структуры данных

Структуры таблиц лексем и идентификаторов данных языка, используемых для хранения, представлены в приложении А.

В таблице лексем содержатся сами лексемы, строка для каждой лексемы, в которой она была замечена. Так же размер самой таблицы лексем В таблице идентификаторов содержится имя идентификатора, его номер в таблице лексем, тип данных, смысловой тип идентификатора и его значение, а также имя родительской функции.

## 3.6 Структура и перечень сообщений лексического анализатора

Перечень сообщений лексического анализатора представлен в таблице 3.2.

Таблица 3.2 Список ошибок лексического анализатора

|  |  |
| --- | --- |
| Описание ошибки | Код ошибки |
| Превышен максимальный размер таблицы лексем | 120 |
| Слишком большое имя переменной или недопустимое имя переменной | 121 |
| Таблица идентификаторов переполнена | 122 |
| Нераспознанная лексема | 123 |
| Нарушен формат строкового литерала | 124 |
| Перезапись идентификатора | 125 |
| Неизвестная переменная | 126 |
| Превышена длина строкового литерала | 127 |
| Не удалось создать файл с таблицей лексем или идентификаторов | 128 |
| Повторное объявление main | 129 |
| Превышена длина символьного литерала | 130 |
| Отсутствие входной точки программы | 131 |
| Недопустимая операция над данным типом | 143 |

## 3.7 Принцип обработки ошибок

При возникновении ошибки работа транслятора прекращается. По возможности, записывается строка и столбец положения ошибки в исходном коде программы.

Запись информации об ошибке осуществляется либо в файл протокола работа лексического анализатора, либо выводится в консоль.

## 3.8 Параметры лексического анализатора

Входным параметром лексического анализатора является исходный текст программы, написанный на языке BKV-2022.

## 3.9 Алгоритм лексического анализа

Последовательность выполнения алгоритма работы лексического анализатора представлена ниже.

1. Считывание текста из исходного файла в буфер;
2. Далее – посимвольное считывание из буфера. Если текущий символ - конец строки – пункт 8;
3. Проверка символа на допустимость;
4. Если текущий символ сепаратор – запись в таблицу лексем. Если нет – дозапись в строку-буфер до тех пор, пока сепаратор не встретится;
5. Проверка строки-буфера на одно из зарезервированных слов. Если такое есть – запись в таблицу лексем и переход к пункту 2;
6. Проверка строки-буфера на соответствие одному из конечных автоматов. Если нет – пункт 8, выход из программы с ошибкой;
7. Запись в таблицу лексем соответствующего значения. Запись в таблицу идентификаторов того или иного идентификатора или литерала. Переход к пункту 2;
8. Конец работы анализатора.

Все регулярные выражения представлены в приложении А.

## 3.10 Контрольный пример

Результат работы – таблицы лексем и идентификаторов – представлены 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, полученная после фазы лексического анализа. Выходом – дерево разбора.

## 4.2 Контекстно-свободная грамматика, описывающая синтаксис

В синтаксическом анализаторе транслятора языка BKV-2022 используется контекстно-свободная грамматика, где

* T – множество терминальных символов (описано в разделе 1.2),
* 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 ослабленную нормальную форму Грейбах:

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

Программное описание правил языка BKV-2022 представлено в приложении Б.

Перечень правил, составляющих грамматику языка, и описание нетерминальных символов BKV-2022 представлены ниже в таблице 4.1.

Таблица 4.1 – Грамматика языка BKV-2022

|  |  |
| --- | --- |
| Нетерминал | Цепочки правил |
| S | tfi(F){rE;};S | tfi(F){NrE;};S | i{NrE;}; |
| N | ti; |  ti;N |  rE; |  i = E; |  i=E;N |  tfi(F); |  tfi(F);N |  oE; |  oE;N |  ti = E;|  ti = E;N |  c(EC){N}N |  c(EC){N}|  i = (t)i;|  i = (t)i;N |  i = (t)E; |  i = (t)E;N |
| E | i | l | (E) | (EC) | i(W) | iM | lM | (E)M | i(W)M | i() | i()M |
| W | i | l | i,W | l,W |
| F | ti | ti,F |
| С | <E | >E | ~E | !E |
| M | vE | vEM |

## 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 BKV-2022. Данные структуры представлены в приложении В

## 4.5 Описание алгоритма синтаксического разбора

Принцип работы конечного автомата с магазинной памятью представлен ниже.

1)В магазин записывается стартовый символ.

2)На основе полученной ранее таблицы лексем формируется входная лента.

3)Запускается автомат.

4)Выбирается цепочка по первому символу, 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 магазина. Иначе возвращаемся в предыдущее сохраненное состояние и выбираем другое правило нетерминала.

6)Если наш символ достиг дна стека, и лента в этот момент пуста, то синтаксический анализ выполнен успешно. Иначе генерируется исключение.

## 4.6 Структура и перечень сообщений синтаксического анализатора

Перечень сообщений синтаксического анализатора представлен в таблице 4.3.

Таблица 4.3 Список ошибок синтаксического анализатора

|  |  |
| --- | --- |
| Описание ошибки | Код ошибки |
| Неверная структура программы | 600 |
| Ошибочный оператор | 601 |
| Ошибка в выражении | 602 |
| Ошибка в параметрах функции | 603 |
| Ошибка в параметрах вызываемой функции | 604 |
| Ошибочное определение логического условия | 605 |

## 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 и правила разбора, которые записываются в файл протокола данного этапа обработки.

## 4.8 Принцип обработки ошибок

Обработка ошибок происходит следующим образом:

* синтаксический анализатор перебирает все правила и цепочки правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем;
* если невозможно подобрать подходящую цепочку, то генерируется соответствующая ошибка, которая записывается в протокол работы и программа останавливается.

## 4.9 Контрольный пример

Пример разбора синтаксическим анализатором исходного кода на языке BKV-2022 представлен в приложении Г. Дерево разбора исходного кода также представлено в приложении Г.

**5. Разработка семантического анализатора**

## 5.1 Структура семантического анализатора

Часть ошибок семантического анализа обрабатываются на этапе лексического анализа. Но ошибки, требующие более сложной обработки (например, несоответствие типов операндов) вынесены в отдельный этап, следующий после синтаксического анализа.

## 5.2 Функции семантического анализатора

Семантический анализатор выполняет проверку на основные правила языка (семантики языка), которые описаны в разделе 1.16.

## 5.3 Перечень сообщений семантического анализатора

Сообщения, формируемые семантическим анализатором, представлены ниже, в таблице 5.1.

Таблица 5.1 - Список ошибок лексического анализатора

|  |  |
| --- | --- |
| Описание ошибки | Код ошибки |
| Имя идентификатора не может быть ключевым словом | 132 |
| Тип функции и тип возвращаемого значения отличаются | 133 |
| Несоответствие присваиваемого типа типу переменной | 134 |
| Запрещено присваивать значение функции | 136 |
| Неверное количество параметров вызываемой функции | 137 |
| Несоответствие параметров в вызываемой функции | 138 |
| Функция strl должна принимать параметр типа string | 139 |
| Функция cmpr должна принимать параметр типа string | 140 |
| Функция ничего не возвращает | 141 |
| Нарушена структура программного блока | 142 |
| Недопустимое значение переменной типа INT | 146 |
| Недопустимое значение переменной типа CHAR | 147 |
| Нарушен формат логического выражения | 148 |

## 5.4 Принцип обработки ошибок

При возникновении ошибки работа транслятора прекращается. По возможности, записывается строка и столбец положения ошибки в исходном коде программы.

Запись информации об ошибке осуществляется либо в файл протокола работа лексического анализатора, либо выводится в консоль.

## 5.5 Контрольный пример

Результат работы контрольного примера представлены в таблице 5.2.

Таблица 5.2 – Результат работы семантического анализатора

|  |  |
| --- | --- |
| Фрагмент программы | Сообщение об ошибке |
| bool function lengthSum(string a, string b)  {  int k =strl(a) + strl(b);  return k;  }; | Ошибка 133:  [Семантическая] Тип функции и тип возвращаемого значения отличаются |
| …  y = 'c'+ 4;  … | Ошибка 134:  [Семантическая] Несоответствие присваиваемого типа типу переменной |
| int function multiply(int x, int y)  {  return (x \* y);  };  main  {  Char k;  k = multiply(4,3);  … | Ошибка 134:  [Семантическая] Несоответствие присваиваемого типа типу переменной |
| int function multiply(int x, int y)  {  return (x \* y);  };  …  multiply =… | Ошибка 136:  [Семантическая] Запрещено присваивать значение функции |
| …  int f = 4294967296;  … | Ошибка 146:  [Семантическая] Недопустимое значение переменной типа INT |

# 6. Преобразование выражений

## 6.1 Выражения, допускаемые языком

В языке BKV-2022 допускаются выражения, применимые к целочисленным типам данных, в так же выражения условий. В выражениях поддерживаются арифметические операции, такие как +, -, \*, / и (), и вызовы функций как операнды арифметических выражений. Для условных выражений те же операции, но с обязательным использованием знаков > , <, != или == (проверка на равенство, не присваивание), эти операции не имеют приоритета и записываются в таблицу лексем по мере их встречи в исходном файле.

Приоритет операций представлен в таблице 6.1.

Таблица 6.1 – Приоритет операций в языке BKV-2022

|  |  |
| --- | --- |
| Приоритет | Операция |
| 1 | ( |
| 1 | ) |
| 2 | , |
| 3 | + |
| 3 | - |
| 4 | \* |
| 4 | / |
| 5 | ( – скобка параметров функции |
| 5 | ) – скобка параметров функции |

## 6.2 Польская запись

Выражения в языке BKV-2022 не преобразовываются к обратной польской записи.

# 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.

## 7.2 Представление типов данных в оперативной памяти

Элементы таблицы идентификаторов расположены в разных сегментах языка ассемблера – .data и .const. Идентификаторы языка размещены в сегменте данных(.data). Литералы – в сегменте констант (.const). Соответствия между типами данных идентификаторов на языке BKV-2022 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка BKV-2022 и языка Ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке BKV-2022 | Тип идентификатора на языке ассемблера | Пояснение |
| int | SDWORD | Хранит целочисленный тип данных со знаком. |
| char | BYTE | Хранит код символа в кодировке ASCII |
| string | BYTE – для строковых литералов  DWORD – для переменных | Литерал хранит последовательность байтов.  Переменные хранят указатель на начало строки литерала. |
| bool | BYTE | Хранит целочисленный тип данных, принимающий в качестве значения 0 или 1. |

## 7.3 Алгоритм работы генератора кода

При разборе таблицы лексем происходит преобразование лексем в соответствующий код на ассемблере. При встрече определённой лексемы происходит генерация соответствующей ей инструкции или выражения.

На листинге 7.1 представлен пример описания лексемы на языке Ассемблера.

**if** (str == "o")

{

**int** l = i;

**while** (lexTable.table[l].lexema != LEX\_SEMICOLON)

        {

            tempEntries.push\_back(lexTable.table[l]);

        l++;

    }

  generateExpression(tempEntries, CS, -1, idTable, idTable.table[lexTable.table[i].idxTI].iddatatype);

CS.add(generateInstructions(PRNT, -1, idTable, idTable.table[lexTable.table[i].idxTI].iddatatype, func\_name));

**continue**;

}

Листинг 7.1 – Алгоритм для лексемы ‘o’

Генерируемый код записывается в файл c расширением .asm. Сгенерированный код можно посмотреть в приложении Е.

# 8. Тестирование транслятора

## 8.1 Тестирование контрольного примера

В результате обработки исходного кода программы на языке BKV-2022 формируется протокол работы транслятора, содержащий основную информацию о ходе работы транслятора и ошибках в случае их возникновения.

В результате обработки транслятором исходного кода программы, представленного в приложении А, формируется информирование об ошибках в общем протоколе работы.

## 8.2 Тестирование программ с ошибками

Транслятор языка BKV-2022 осуществляет диагностику и выявление ошибок на разных этапах трансляции. Ниже будут приведены результаты обработки транслятором исходного кода с заранее допущенными ошибками.

Тестирование ошибок на этапе лексического анализатора представлено в листинге 8.1 и рисунке 8.1.

|  |
| --- |
| main  {  int number = 3;  char number;  return number;  } |

Листинг 8.1 – Код с лексической ошибкой

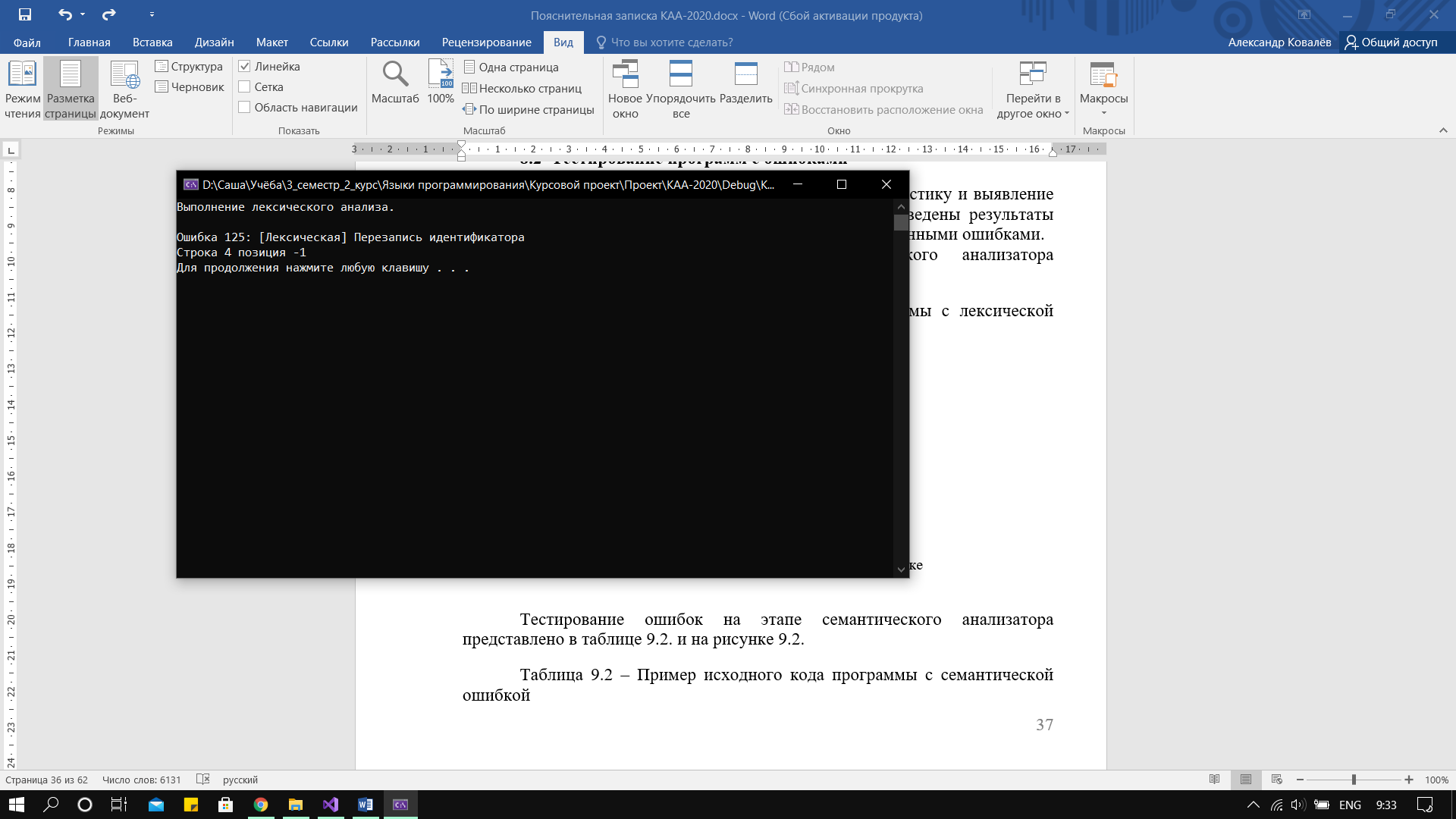


Рисунок 8.1 – Сообщение о допущенной лексической ошибке

Тестирование ошибок на этапе семантического анализатора представлено в листинге 8.2. и на рисунке 8.2.

|  |
| --- |
| int function fun(int a)  {  Int result  result = a  result = result+a  return result;  };  main  {  Int a = 5  Char f = ‘f’  fun = 4;  out fun;  return 0;  }; |

Листинг 8.2 – Код с семантической ошибкой

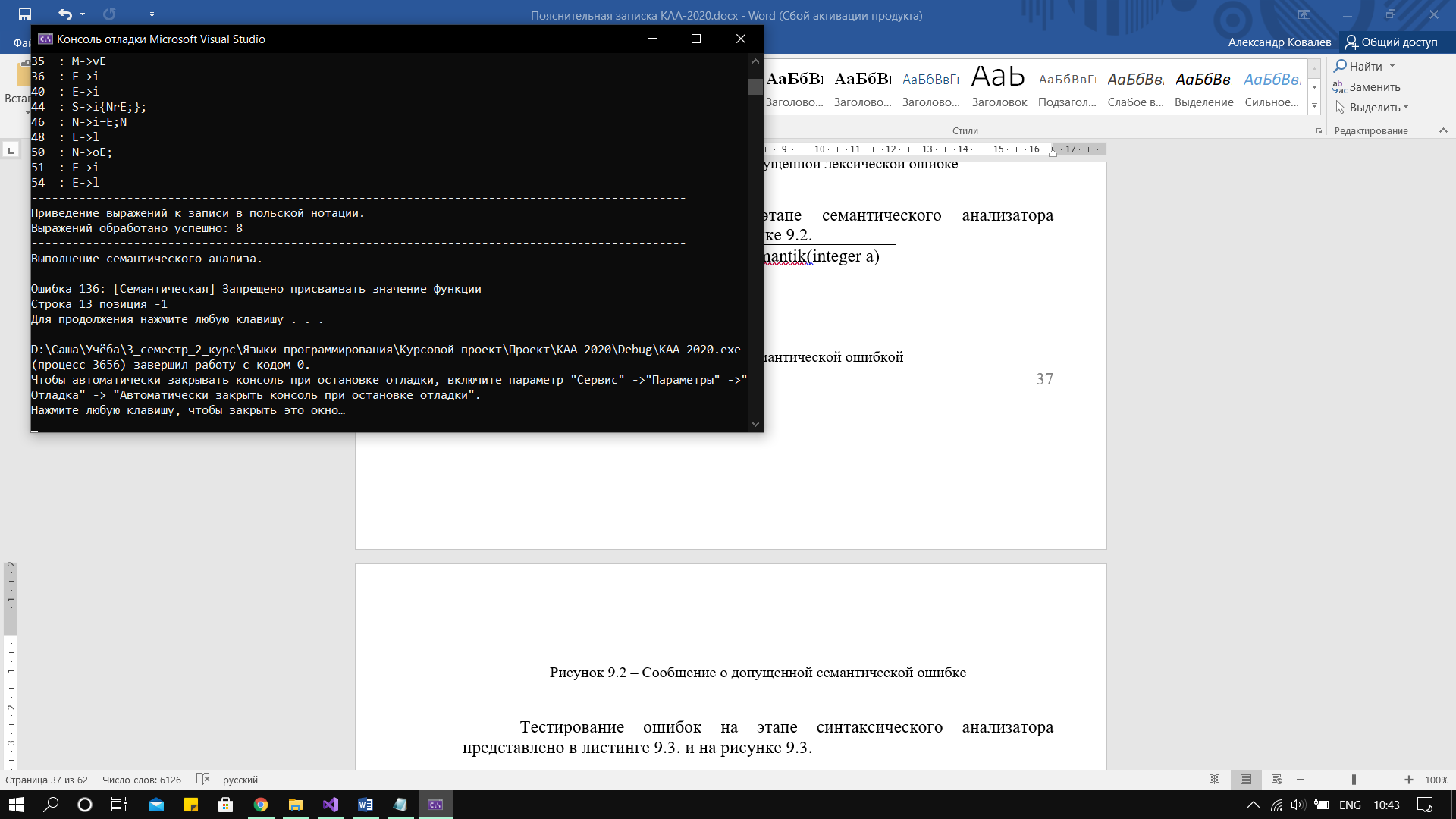


Рисунок 8.2 – Сообщение о допущенной семантической ошибке

Тестирование ошибок на этапе синтаксического анализатора представлено в листинге 8.3. и на рисунке 8.3.

|  |
| --- |
| main  {  int number = 0;  int iter = 2;  if(iter + number)  {  number = iter /2;  out "I don't work";  }  return number;  }; |

Листинг 8.3. – Код с синтаксической ошибкой

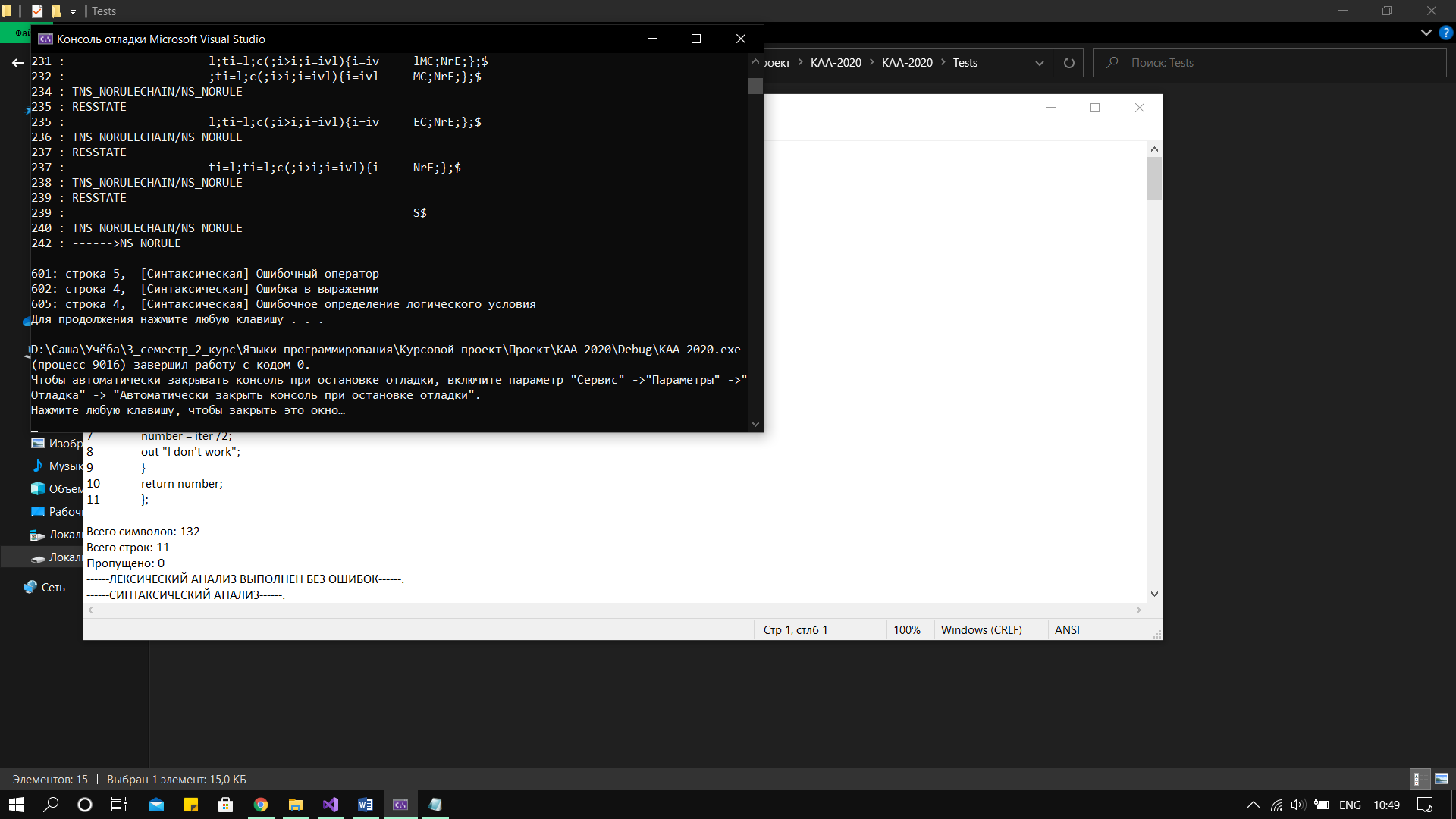


Рисунок 8.3. – Сообщение о допущенной синтаксической ошибке

## Заключение

В данном курсовом проекте реализованы поставленные минимальные требования, а также дополнен определённый функционал исходного результата работы. Основной целью курсовой работы было понять принцип работы языков программирования и усвоить эти знания на собственном примере. В ходе работы было изучено много нового, а также закреплены знания, которые были получены ранее. Данный курсовой проект позволил углубиться в изучение таких языков как С++ и ассемблер. При написании приложения были усвоены такие понятия как синтаксический, лексический и семантический анализатор и многие другие.

В итоге был получен примитивный язык программирования BKV-2022, содержащий простые конструкции, позволяющие выполнять несложные математические операции.

Окончательная версия языка включает:

* 4 типа данных;
* Поддержку операции вывода;
* Возможность вызова функций стандартной библиотеки;
* Наличие 4 арифметических операторов для вычисления выражений;
* Наличие одной директивы препроцессора;
* Возможность построения вложенных циклов;
* Поддержку перегрузку функций;
* Структурированную систему для обработки ошибок пользователя.

Основные характеристики транслятора BKV-2022:

* Возможность обработки до 3 входных параметров;
* Возможность обработки ошибок;
* Реализация 17 конечных автоматов;
* Реализация 45 цепочек правил грамматики;
* Наличие более 2000 строк кода.

Полученные знания при выполнении курсового проекта будут способствовать углубленному изучению новых технологий и языков программирования.

# Список использованных источников

1. Программирование на языке ассемблера [Электронный ресурс] - Режим доступа: <http://natalia.appmat.ru/c&c++/assembler.html>

2. Кодировка ASCII > Таблица символов 1251 (ANSI, WIN) [Электронный ресурс] - Режим доступа: <https://tools.otzyvmarketing.ru/blog/poleznoe/Osnova-osnov-kodirovka-ASCII-i-ee-sovremennye-interpretacii->

3. Макросредства языка Ассемблер [Электронный ресурс] - Режим доступа: http://mf.grsu.by/UchProc/livak/arxiv\_22102010/kursi/programming/lections/assm/macros.htm

4. Парадигмы программирования [Электронный ресурс] – Режим доступа:

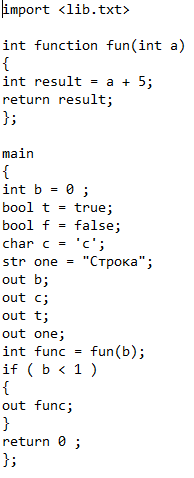
<http://progopedia.ru/paradigm/>

5. Язык программирования. Языки низкого и высокого уровня [Электронный ресурс] – Режим доступа: https://unetway.com/blog/azyk-programmirovania-azyki-nizkogo-i-vysokogo-urovna

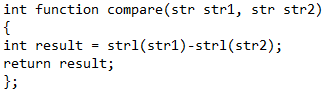
# Приложения

## Контрольный пример

Содержимое файла in.txt:



Содержимое файла lib.txt:



## Приложение А

Таблица лексем:

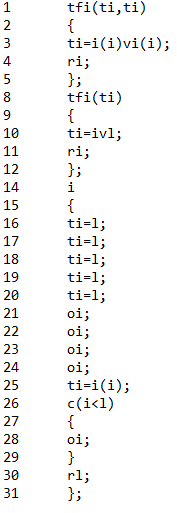
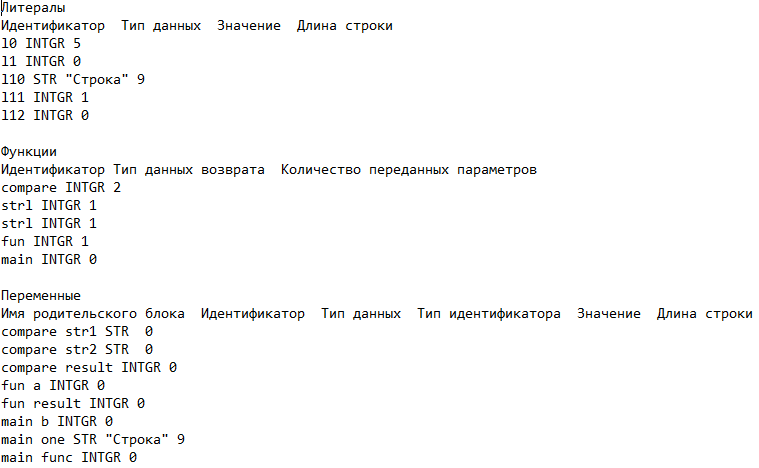
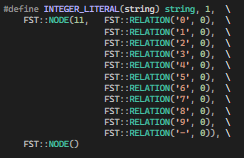
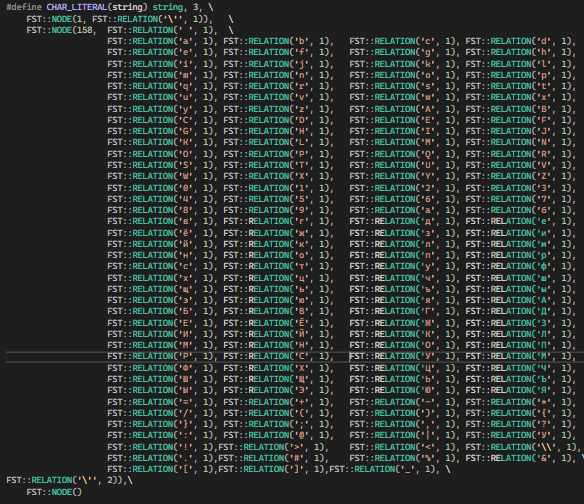
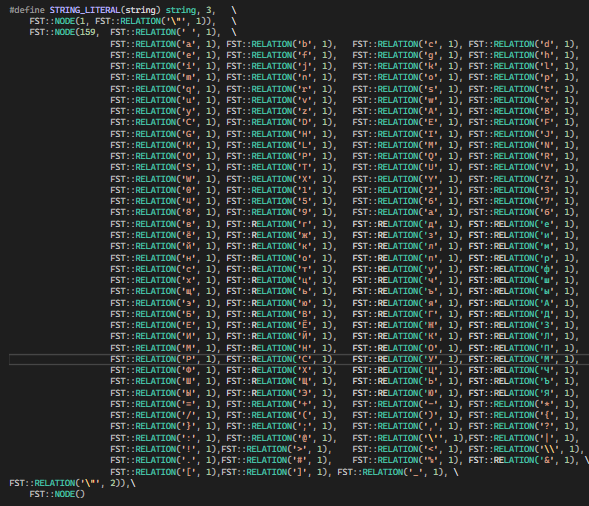
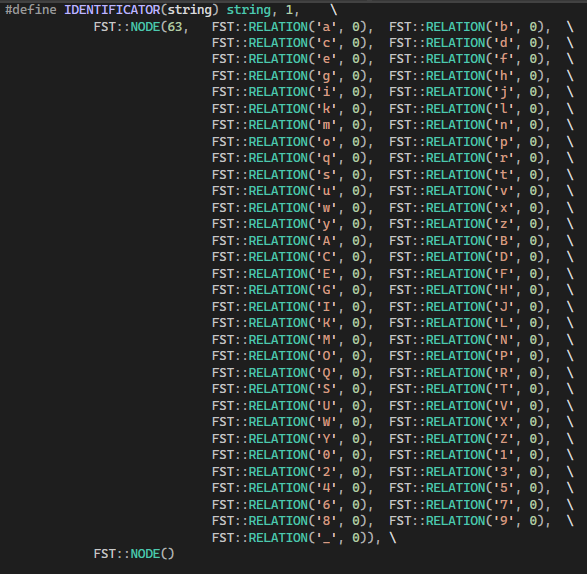
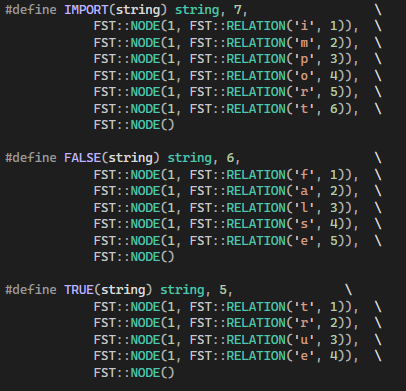
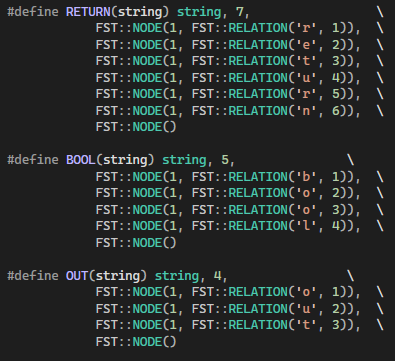
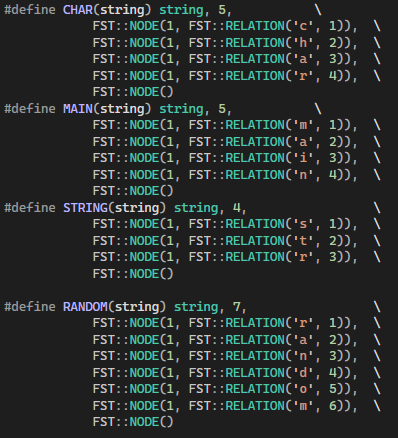
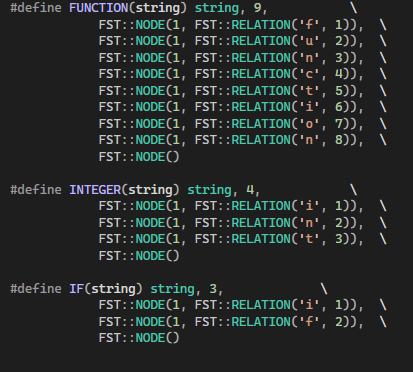


Таблица идентификаторов:

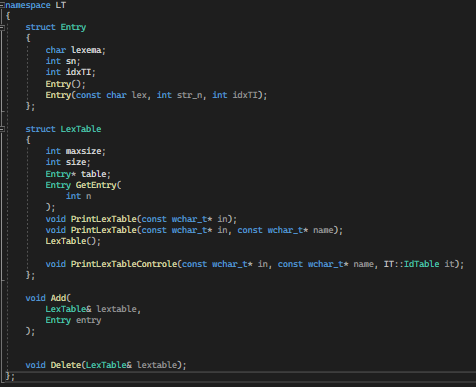


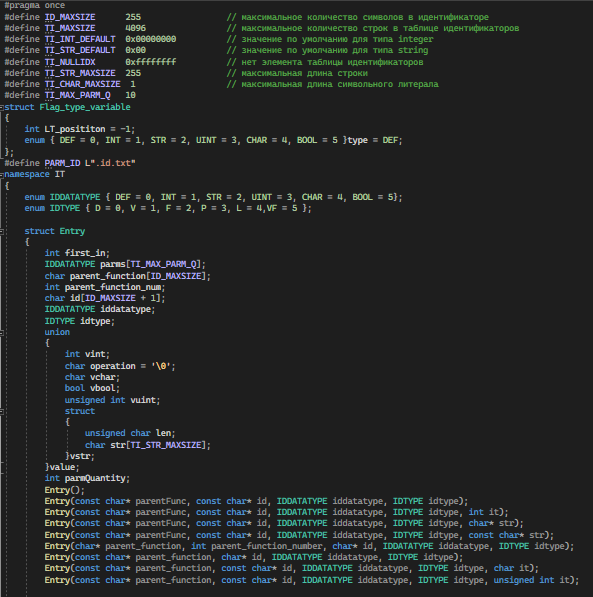
Экземпляры конечных автоматов для распознавания лексических конструкций языка:

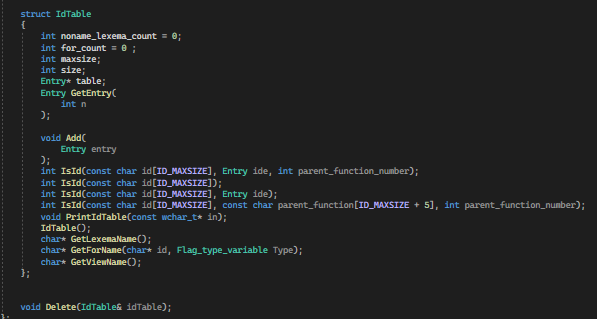




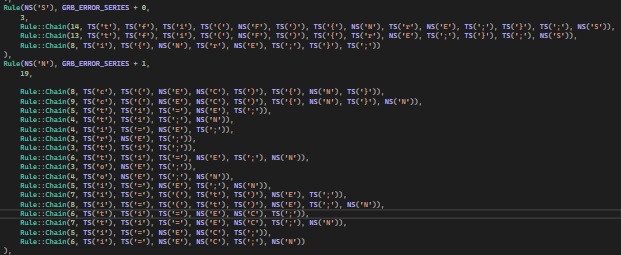


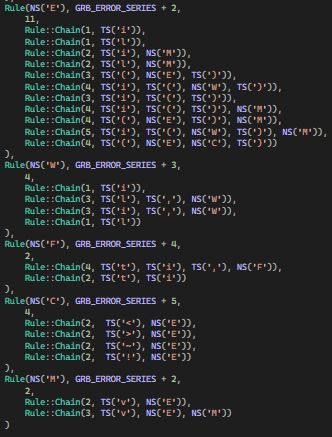




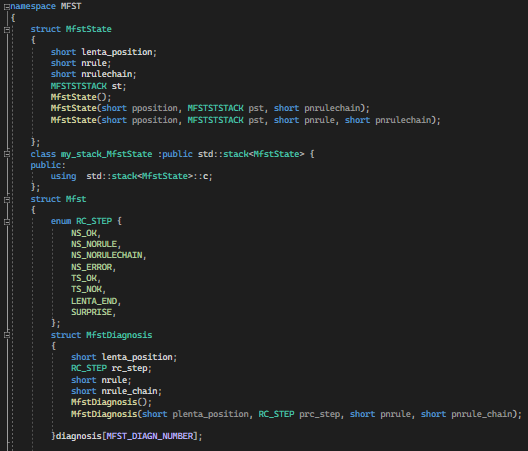
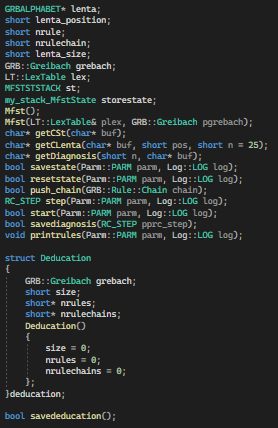


## Приложение Б





## Приложение В

## Приложение Г

Начало разбора

Шаг : Правило Входная лента Cтек

0 : S->tfi(F){NrE;};S S$

0 : SAVESTATE: 1

0 : tfi(F){NrE;};S$

1 : fi(ti,ti){ti=i(i)vi(i);ri fi(F){NrE;};S$

2 : i(ti,ti){ti=i(i)vi(i);ri; i(F){NrE;};S$

3 : (ti,ti){ti=i(i)vi(i);ri;} (F){NrE;};S$

4 : ti,ti){ti=i(i)vi(i);ri;}; F){NrE;};S$

5 : F->ti,F ti,ti){ti=i(i)vi(i);ri;}; F){NrE;};S$

5 : SAVESTATE: 2

5 : ti,ti){ti=i(i)vi(i);ri;}; ti,F){NrE;};S$

6 : i,ti){ti=i(i)vi(i);ri;};t i,F){NrE;};S$

7 : ,ti){ti=i(i)vi(i);ri;};tf ,F){NrE;};S$

8 : ti){ti=i(i)vi(i);ri;};tfi F){NrE;};S$

9 : F->ti,F ti){ti=i(i)vi(i);ri;};tfi F){NrE;};S$

9 : SAVESTATE: 3

Конец разбора:

433 : i;}rl;};;;;;;;;;;;;;;;;;; E;}rE;};$

434 : E->i i;}rl;};;;;;;;;;;;;;;;;;; E;}rE;};$

434 : SAVESTATE: 45

434 : i;}rl;};;;;;;;;;;;;;;;;;; i;}rE;};$

435 : ;}rl;};;;;;;;;;;;;;;;;;;; ;}rE;};$

436 : }rl;};;;;;;;;;;;;;;;;;;;; }rE;};$

437 : rl;};;;;;;;;;;;;;;;;;;;;; rE;};$

438 : l;};;;;;;;;;;;;;;;;;;;;;; E;};$

439 : E->l l;};;;;;;;;;;;;;;;;;;;;;; E;};$

439 : SAVESTATE: 46

439 : l;};;;;;;;;;;;;;;;;;;;;;; l;};$

440 : ;};;;;;;;;;;;;;;;;;;;;;;; ;};$

441 : };;;;;;;;;;;;;;;;;;;;;;;; };$

442 : ;;;;;;;;;;;;;;;;;;;;;;;;; ;$

443 : ;;;;;;;;;;;;;;;;;;;;;;;;; $

444 : 6

445 : ------END

Правила разбора

0 : S->tfi(F){NrE;};

S 4 : F->ti,F

7 : F->ti

11 : N->ti=E;

14 : E->i(W)M

16 : W->i

18 : M->vE

19 : E->i(W)

21 : W->i

25 : E->i

29 : S->tfi(F){NrE;};S

33 : F->ti

37 : N->ti=E;

40 : E->iM

41 : M->vE

42 : E->l

45 : E->i

49 : S->i{NrE;};

51 : N->ti=E;N

54 : E->l

56 : N->ti=E;N

59 : E->l

61 : N->ti=E;N

64 : E->l

66 : N->ti=E;N

69 : E->l

71 : N->ti=E;N

74 : E->l

76 : N->oE;N

77 : E->i

79 : N->oE;N

80 : E->i

82 : N->oE;N

83 : E->i

85 : N->oE;N

86 : E->i

88 : N->ti=E;N

91 : E->i(W)

93 : W->i

96 : N->c(EC){N}

98 : E->i

99 : C-><E

100 : E->l

103 : N->oE;

104 : E->i

108 : E->l

## Приложение Д

Код на языке ассемблера:

.586

.model flat, stdcall

includelib userlib.lib

includelib kernel32.lib

includelib libucrt.lib

ExitProcess PROTO : DWORD

strl PROTO : DWORD

random PROTO : DWORD

outstr PROTO : DWORD

outbool PROTO : DWORD

outchar PROTO : DWORD

outuint PROTO : DWORD

outint PROTO : SDWORD

strcon PROTO : DWORD, : DWORD

intToUint PROTO : SDWORD

uintToInt PROTO : DWORD

.stack 4096

.const

overflow db 'ERROR: VARIABLE OVERFLOW', 0

null\_division db 'ERROR: DIVISION BY ZERO', 0

l0 SDWORD 5

l1 SDWORD 0

l2 BYTE 1

l3 BYTE 0

l4 BYTE 99

l10 BYTE "Строка", 0

l11 SDWORD 1

l12 SDWORD 0

.data

compareresult SDWORD 0

funresult SDWORD 0

mainb SDWORD 0

maint BYTE 0

mainf BYTE 0

mainc BYTE 0

mainone DWORD ?

mainfunc SDWORD 0

.code

compare22\_proc PROC, comparestr1 : DWORD, comparestr2 : DWORD

pop ebx

pop eax

sub eax, ebx

jo EXIT\_OVERFLOW

push eax

pop compareresult

push compareresult

jmp EXIT

EXIT\_DIV\_ON\_NULL:

push offset null\_division

call outstr

push - 1

call ExitProcess

EXIT\_OVERFLOW:

push offset overflow

call outstr

push - 2

call ExitProcess

EXIT:

pop eax

ret 8

compare22\_proc ENDP

fun1\_proc PROC, funa : SDWORD

push funa

pop eax

pop ebx

add eax, ebx

jo EXIT\_OVERFLOW

push eax

push l0

pop funresult

push funresult

jmp EXIT

EXIT\_DIV\_ON\_NULL:

push offset null\_division

call outstr

push - 1

call ExitProcess

EXIT\_OVERFLOW:

push offset overflow

call outstr

push - 2

call ExitProcess

EXIT:

pop eax

ret 4

fun1\_proc ENDP

main PROC

push l1

pop mainb

push offset l2

pop dword ptr maint

push offset l3

pop dword ptr mainf

push offset l4

pop dword ptr mainc

mov mainone,offset l10

push mainb

call outint

push dword ptr mainc

call outchar

push offset maint

call outbool

push dword ptr mainone

call outstr

pop mainfunc

push l11

pop mainb

metkal11:

push mainfunc

call outint

pop l11

push mainb

pop ebx

pop eax

cmp ebx, eax

jb metkal11

jmp EXIT

EXIT\_DIV\_ON\_NULL:

push offset null\_division

call outstr

push - 1

call ExitProcess

EXIT\_OVERFLOW:

push offset overflow

call outstr

push - 2

call ExitProcess

EXIT:

push 0

call ExitProcess

main ENDP

end main