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 KNS-2022»

Выполнил студент Каребо Никита Сергеевич

(Ф.И.О.)

Руководитель проекта ст.пр. Наркевич Аделина Сергеевна

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

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

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

Консультанты ст.пр. Наркевич Аделина Сергеевна

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

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

Нормоконтролер ст.пр. Наркевич Аделина Сергеевна

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

Курсовой проект защищен с оценкой

Минск 2022

**Содержание**

[**Введение** 5](#_Toc122349858)

[**Глава 1. Спецификация языка программирования** 6](#_Toc122349859)

[**1.1. Характеристика языка программирования** 6](#_Toc122349860)

[**1.2. Алфавит языка** 6](#_Toc122349861)

[**1.3. Символы сепараторы** 6](#_Toc122349862)

[**1.4. Применяемые кодировки** 7](#_Toc122349863)

[**1.5. Типы данных** 7](#_Toc122349864)

[**1.6. Преобразование типов данных** 8](#_Toc122349865)

[**1.7. Идентификаторы** 9](#_Toc122349866)

[**1.8. Литералы** 9](#_Toc122349867)

[**1.9. Объявление данных** 9](#_Toc122349868)

[**1.10. Инициализация данных** 10](#_Toc122349869)

[**1.11. Инструкции языка** 10](#_Toc122349870)

[**1.12. Операции языка** 11](#_Toc122349871)

[**1.13. Выражения и их вычисления** 11](#_Toc122349872)

[**1.14. Программные конструкции языка** 12](#_Toc122349873)

[**1.15. Область видимости** 12](#_Toc122349874)

[**1.16. Семантические проверки** 13](#_Toc122349875)

[**1.17. Распределение оперативной памяти на этапе выполнения** 13](#_Toc122349876)

[**1.18. Стандартная библиотека и её состав** 13](#_Toc122349877)

[**1.19. Ввод и вывод данных** 14](#_Toc122349878)

[**1.20. Точка входа** 14](#_Toc122349879)

[**1.21. Препроцессор** 14](#_Toc122349880)

[**1.22. Соглашения о вызовах** 14](#_Toc122349881)

[**1.23. Объектный код** 14](#_Toc122349882)

[**1.24. Классификация сообщений транслятора** 15](#_Toc122349883)

[**1.25. Контрольный пример** 15](#_Toc122349884)

[**Глава 2. Структура транслятора** 16](#_Toc122349885)

[**2.1. Компоненты транслятора, их назначение и принципы взаимодействия** 16](#_Toc122349886)

[**2.2. Перечень входных параметров транслятора** 17](#_Toc122349887)

[**2.3. Перечень протоколов, формируемых транслятором** 17](#_Toc122349888)

[**Глава 3. Разработка лексического анализатора** 18](#_Toc122349889)

[**3.1. Структура лексического анализатора** 18](#_Toc122349890)

[**3.2. Контроль входных символов** 18](#_Toc122349891)

[**3.3. Удаление избыточных символов** 19](#_Toc122349892)

[**3.4. Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов** 19](#_Toc122349893)

[**3.5. Основные структуры данных** 21](#_Toc122349894)

[**3.6. Принцип обработки ошибок** 21](#_Toc122349895)

[**3.7. Структура и перечень сообщений лексического анализатора** 21](#_Toc122349896)

[**3.8. Параметры лексического анализатора** 22](#_Toc122349897)

[**3.9. Алгоритм лексического анализа** 22](#_Toc122349898)

[**3.10. Контрольный пример** 23](#_Toc122349899)

[**Глава 4. Разработка синтаксического анализатора** 24](#_Toc122349900)

[**4.1. Структура синтаксического анализатора** 24](#_Toc122349901)

[**4.2. Контекстно свободная грамматика, описывающая синтаксис языка** 24](#_Toc122349902)

[**4.3. Построение конечного магазинного автомата** 26](#_Toc122349903)

[**4.4. Основные структуры данных** 27](#_Toc122349904)

[**4.5. Описание алгоритма синтаксического разбора** 27](#_Toc122349905)

[**4.6. Структура и перечень сообщений синтаксического анализатора** 28](#_Toc122349906)

[**4.7. Параметры синтаксического анализатора и режимы его работы** 28](#_Toc122349907)

[**4.8. Принцип обработки ошибок** 28](#_Toc122349908)

[**4.9. Контрольный пример** 29](#_Toc122349909)

[**Глава 5. Разработка семантического анализатора** 30](#_Toc122349910)

[**5.1. Структура семантического анализатора** 30](#_Toc122349911)

[**5.2. Функции семантического анализатора** 30](#_Toc122349912)

[**5.3. Структура и перечень сообщений семантического анализатора** 31](#_Toc122349913)

[**5.4. Принцип обработки ошибок** 31](#_Toc122349914)

[**5.5. Контрольный пример** 31](#_Toc122349915)

[**Глава 6. Вычисление выражений** 32](#_Toc122349916)

[**6.1. Выражения, допускаемые языком** 32](#_Toc122349917)

[**6.2. Польская запись и принцип её построения** 32](#_Toc122349918)

[**6.3. Программная реализация обработки выражений** 33](#_Toc122349919)

[**6.4. Контрольный пример** 34](#_Toc122349920)

[**Глава 7. Генерация кода** 35](#_Toc122349921)

[**7.1. Структура генератора кода** 35](#_Toc122349922)

[**7.2. Представление типов данных в оперативной памяти** 35](#_Toc122349923)

[**7.3. Статическая библиотека** 36](#_Toc122349924)

[**7.4.** **Особенности алгоритма генерации кода** 36](#_Toc122349925)

[**7.5. Входные параметры, управляющие генерацией кода** 36](#_Toc122349926)

[**7.6. Контрольный пример** 37](#_Toc122349927)

[**Глава 8. Тестирование транслятора** 38](#_Toc122349928)

[**8.1. Общие положения** 38](#_Toc122349929)

[**8.2. Результаты тестирования** 38](#_Toc122349930)

[**Заключение** 40](#_Toc122349931)

[**Приложения** 41](#_Toc122349932)

[**Приложение А** 41](#_Toc122349933)

[**Приложение Б** 43](#_Toc122349934)

[**Приложение В** 46](#_Toc122349935)

[**Приложение Г** 52](#_Toc122349936)

[**Приложение Д** 57](#_Toc122349937)

[**Приложение Е** 59](#_Toc122349938)

[**Приложение Ж** 61](#_Toc122349939)

[**Приложение З** 63](#_Toc122349940)

[**Приложение И** 65](#_Toc122349941)

[**Список использованной литературы** 66](#_Toc122349942)

# **Введение**

Задачей курсового проекта является разработка транслятора для своего языка программирования: KNS-2022. Предназначен этот язык программирования для выполнения простейших операций над строками и числами.

Транслятор – это комплекс отдельных программ, позволяющих преобразовывать исходный код на одном языке программирования в исходный код на другом языке программирования. Главной задачей моего транслятора является преобразование программы на языке программирования KNS-2022, в код понятный компьютеру, а точнее в код языка ассемблера.

Язык ассемблера — машинно-ориентированный язык программирования низкого уровня. Представляет собой систему обозначений, используемую для представления в удобно читаемой форме программ, записанных в машинном коде. Его команды прямо соответствуют отдельным командам машины или их последовательностям.

Исходя из цели курсового проекта, были определены следующие задачи:

– разработка спецификации языка программирования;

– разработка структуры транслятора;

– разработка лексического анализатора;

– разработка синтаксического анализатора;

– разработка семантического анализатора;

– обработка выражений;

– генерация кода на язык Assembler;

– тестирование транслятора.

# **Глава 1. Спецификация языка программирования**

## **Характеристика языка программирования**

Язык программирования KNS–2022 является языком программирования высокого уровня. Этот язык строго типизируемый, то есть в языке отсутствует преобразование типов. Так же язык компилируемый, но не является объектно-ориентированным.

* 1. **Алфавит языка**

Исходный код KNS-2022 может содержать символы латинского алфавита верхнего и малого регистров, цифры десятичной системы счисления от 0 до 9, символы табуляции и перевода строки, спецсимволы: [] () , ; : # + - / \* % & > < !.

## **Символы сепараторы**

Символы сепараторы служат в качестве разделителей операций языка. Сепараторы, используемые в языке программирования KNS-2022, приведены в таблице 1.1.

Таблица 1.1. Символы сепараторы

|  |  |
| --- | --- |
| Символ(ы) | Назначение |
| пробел | Разделитель цепочек, в именах идентификаторов и ключевых слов использовать нельзя |
| **{**  } | Блок функции или цикла |
| **(**  **)** | Блок фактических или формальных параметров функции, а так же приоритет арифметических операций |
| **,** | Разделитель параметров функций |
| **+ - \* /** | Арифметические операции |
| **>, <, !, >=, <=, &** | Операции сравнения: больше, меньше, не равно, больше или равно, меньше или равно |
| **;** | Разделитель программных конструкций |
| **=** | Оператор присваивания |
| **$...$** | Указание родительского блока |
| **‘’** | Запись строкового литерала |

## **Применяемые кодировки**

В основе алфавита KNS-2022 лежит таблица символов Windows-1251. Исходный код KNS-2022 может содержать символы кириллицы, латинского алфавита верхнего и малого регистров, цифры десятичной системы счисления от 0 до 9, символы табуляции и перевода строки, спецсимволы: [] () , ; : # + - / \* % & > < !. Содержимое таблицы Windows-1251 представлено на рисунке 1.1.

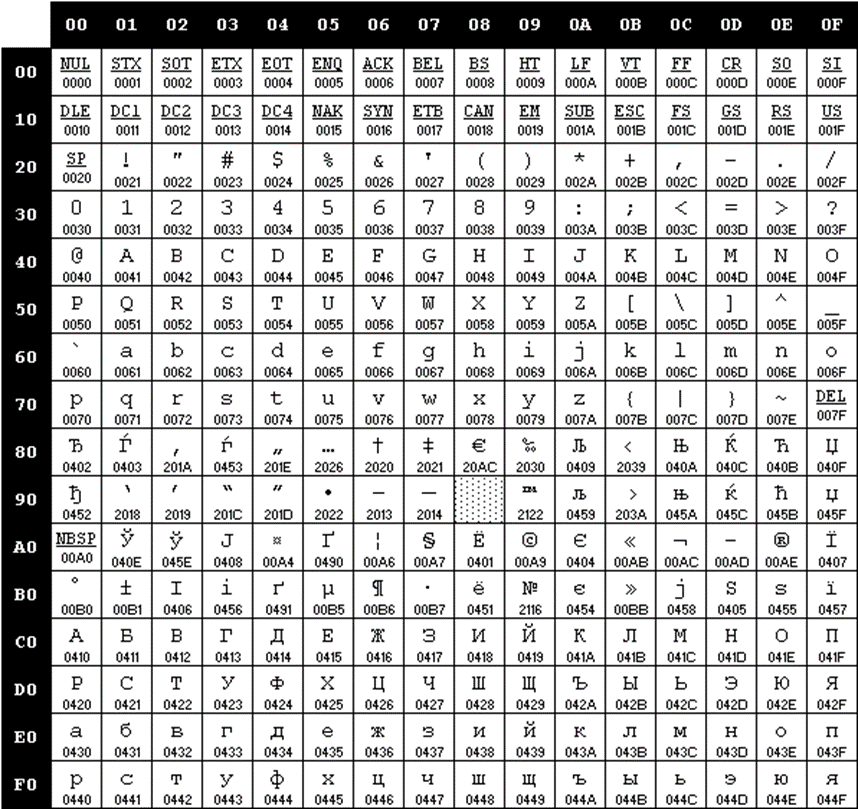


Рисунок 1.1 – Алфавит входных символов

## **Типы данных**

В языке используется 4 основных типов данных, которые описываются в таблице 1.2.

Таблица 1.2. Описание типов данных

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| unint | Фундаментальный тип данных. Предусмотрен для объявления целочисленных, беззнаковых данных (4 байта). Автоматически инициализируется нулевым значением.  Поддерживаемые операции:  + (бинарный) – оператор сложения;  - (бинарный) – оператор вычитания;  \* (бинарный) – оператор умножения;  % (бинарный) – оператор деления;  = (бинарный) – оператор присваивания  В качестве операторов сравнения поддерживаются следующие операторы:  & (бинарный) – оператор “равенство”  > (бинарный) – оператор “больше”;  < (бинарный) – оператор “меньше”  ! (бинарный) – оператор “не равно”  >= (бинарный) – оператор “больше либо равно”  <= (бинарный) – оператор “меньше либо равно” |
| str | Фундаментальный тип данных. Предусмотрен для объявления строк. (1 символ – 1 байт).Максимальный размер строки может быть 254 символа. Данные этого типа заключаются в “”. Автоматическая инициализация: символ конца строки ”\0”. |
| unint  [<литерал>] | Фундаментальный тип данных. Используется для работы с цело-численными массивами. Каждый элемент занимает в памяти 2 бай-та. Инициализация по умолчанию: каждый элемент равен 0. Переменную такого типа можно инициализировать некоторым набором целочисленных литералов внутри блока data {…}; |
| boolean | Фундаментальный тип данных (1 байта), используемый для объявления логической переменной, которая принимает одно из двух значений: true/1 или false/0. Без явно указанной инициализации переменной, присваивается нулевое значение (false). |

## **Преобразование типов данных**

Преобразование типов данных не поддерживается, т.е. язык является строготипизированным.

## **Идентификаторы**

Для именования переменных, параметров и функций всегда используются идентификаторы. Идентификаторы могут содержать символы как нижнего регистра, так и верхнего. Идентификаторы, объявленные внутри функционального блока, получают префикс, идентичный имени функции, внутри которой они объявлены. Идентификаторы не должны совпадать с ключевыми словами. Не предусмотрены зарезервированные идентификаторы.

Идентификатор должен составляться подобным образом:

* состоит из символов латинского алфавита [A-Z|a-z]+ и непосредственно целое число состоящее из множества [0,1,2,...,9].

## **Литералы**

Литерал — это элемент программы, который непосредственно представляет значение.

В любом языке программирования должны быть литералы, исключением язык программирования KNS–2022 не стал. В своём языке программирования существует 3 типа литералов: целого типа, строковые и логические. Краткое описание литералов представлено в таблице 1.3.

Таблица 1.3. Литералы

|  |  |
| --- | --- |
| Тип литерала | Описание |
| Литералы целого типа | Целочисленные литералы. Представляют непосредственно целое число состоящее из множества [0,1,2,...,9].Данные этого типа могут быть представлены в десятичной системе (123) и в восьмеричной (0173). |
| Строковые литералы | Символы, заключённые в ‘ ’ (одинарные кавычки), инициализируются пустой строкой. |
| Логические литералы | true и false являются логическими литералами. true интерпретируется, как 1, а false как 0. Логический тип относится к целым типам. |

## **Объявление данных**

Объявление всех типов данных, включая переменные и функции, в языке программирования KNS-2022 может происходить только внутри специального блока объявления data{…}. В случае, если необходимо областью видимости пе-ременной объявить функцию, необходимо после объявления самой функции объявить эту переменную и после указания типа данных заключить между двух знаков ‘$’ имя функции, внутри которой данная переменная будет использоваться.

## **Инициализация данных**

Способ инициализации переменной: без значения. При объявлении переменной без значения, ей будет присвоено значение по умолчанию. Для unint нулевое значение, для str признак конца строки, для boolean значение false. Так же способы инициализации переменной представлены в таблице 1.4.

Таблица 1.4. Способы инициализации переменной

|  |  |
| --- | --- |
| Вид инициализации | Примечание |
| <тип данных> <идентификатор>; | Автоматическая инициализация: пере-менные типа uint инициализируются ну-лём, переменные типа str – пустой строкой. |
| <идентификатор> = <значение>; | Присваивание переменной значения. |

## **Инструкции языка**

Все инструкции языка KNS–2022 представлены в таблице1.5.

Таблица 1.5. Инструкции языка

|  |  |
| --- | --- |
| Инструкция | Запись |
| Объявление переменной | <тип данных> <идентификатор>; |
| Объявление с указанием области видимости | <тип данных> $идентификатор функции$ идентификатор; |
| Присваивание | <идентификатор> = <значение>|<идентификатор>; |
| Объявление функции | <тип данных> function <идентификатор> (<тип данных> <идентификатор>, …) {…} |
| Блок инструкций | {  …  } |
| Вывод данных | print <выражение> |
| Вызов функций | <идентификатор функции>(<идентификатор | литерал>,…) |
| Оператор цикла | while (<идентификатор>/<литерал>< логическая операция> <идентификатор>/ <литерал>)  {…} |
| Условный оператор | expression (<идентификатор>/<литерал>< логическая операция> <идентификатор>/ <литерал>)  {…} |
| unint [<литерал>] <идентификатор> <литерал>, …. | Объявление массива целочисленных значений |

## **Операции языка**

Операции языка KNS–2022 и их приоритет представлен в таблице 1.6.

Таблица 1.6. Операторы языка KNS–2022

|  |  |
| --- | --- |
| Тип оператора | Оператор |
| Арифметические | **+ -** сложение  - **-** разность  **\* -** умножение  **% -** деление |
| Сравнения | **>** - больше  **<** - меньше  **!** – не равно  >= – больше или равно  <= - меньше или равно  & - равно |

## **Выражения и их вычисления**

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

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

Перед генерацией кода каждое выражение приводится к записи в польской записи для удобства дальнейшего вычисления выражения на языке ассемблер

## **Программные конструкции языка**

Программные конструкции языка программирования KNS–2022 представлены в таблице 1.7.

Таблица 1.7. Программные конструкции языка KNS–2022

|  |  |
| --- | --- |
| Главная функция (точка входа в приложение) | source  {…} |
| Блок с данными | data  {….} |
| Функция | <тип данных> function <идентификатор> (<тип данных> <идентификатор>, …)  {  …  return <выражение>;  } |
| Цикл | while (<условие>)  {…} |
| Условная конструкция condition | expression (<условие>)  {…} |

## **Область видимости**

Областью видимости идентификатора по умолчанию является функция sourse. В случае, если идентификатор принадлежит какой-либо другой функции, имя которой будет заключено между знаками “$” после указания типа данных, область видимости переменной будет только внутри этой функции.

## **Семантические проверки**

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

Таблица 1.8 – Семантические проверки

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Идентификаторы не должны повторяться |
| 2 | Тип данных передаваемых значений в функцию должен совпадать с типом параметров при её объявлении |
| 3 | В функцию должны быть переданы параметры |
| 4 | Тип данных результата выражения должен совпадать с типом данных идентификатора, которому оно присваивается |
| 5 | Индекс вызываемого элемента массива не должен превосходить размер массива |
| 6 | Функция sourse обязательно должна присутствовать и быть единственной |
| 7 | Блок объявления data обязательно должен присутствовать |
| 8 | Параметру функции нельзя присвоить значение |

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

Все переменные хранятся в куче. Для таблиц лексем и идентификаторов выделяется динамическая память, которая очищается по завершению работы транслятора.

## **Стандартная библиотека и её состав**

Функции стандартной библиотеки реализованы на языке программирования C++, а так же представлены с описанием в таблице 1.9.

Таблица 1.9. Стандартные библиотеки

|  |  |  |  |
| --- | --- | --- | --- |
| Имя функции | Возвращаемое значение | Принимаемые параметры | Описание |
| void print | - | str value | Функция выводит в консоль строковый литерал или переменную, переданный ей через параметры |
| void printcurrentdate | - | - | Функция выводит в консоль текущую дату |
| void printint | - | unint | Функция выводит в консоль целочисленный литерал или переменную, переданный ей через параметры |
| void printcurrenttime | - | uint | Функция выводит в консоль текущую время |
| Void cat1() | - | str dest, str par1 | Функция выполняет присваивание строковой переменной строкового значения |
| Void cat2() | - | str destina-tion, str left\_operand, str right\_operand | Функция выполняет конкатенацию двух строковых значений |

## **Ввод и вывод данных**

В языке KNS-2022 не реализованы средства ввода данных.

Для вывода данных в стандартный поток вывода предусмотрена функция print, в зависимости от принимаемого значения которой вызывается она из соответствующих функций стандартной библиотеки описаны в таблице 1.9.

## **Точка входа**

Каждый исходный код на языке программирования KNS–2022 должен содержать главную функцию (точку входа) source. Повторение функции source не предусмотрено.

## **Препроцессор**

Наличие препроцессора в языке программирования KNS–2022 не предусмотрено.

## **Соглашения о вызовах**

Вызов любых функций происходит при помощи соглашения о вызовах stdcall. Особенности stdcall:

– все параметры функции передаются через стек;

– память высвобождает вызываемый код;

– занесение в стек параметров идёт справа налево.

## **Объектный код**

Язык программирования KNS–2022 транслируется в язык ассемблера.

## **Классификация сообщений транслятора**

При возникновении ошибки в коде программы на языке KNS–2022, транслятором генерируется исключение, которое укажет примерную причину неработоспособности исходного кода. Их классификация представлена в таблице 1.10.

Таблица 1.10. Классификация сообщений транслятора

|  |  |
| --- | --- |
| Интервал | Описание ошибок |
| 0-113 | Системные ошибки, ошибки параметров |
| 119-200 | Ошибки лексического анализа |
| 200-300 | Ошибки синтаксического анализа |
| 300-400 | Ошибки семантического анализа |

## **Контрольный пример**

Контрольный пример представлен во главе Приложения [A](#_Приложение_А).

# **Глава 2. Структура транслятора**

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

Транслятор преобразует программу, написанную на языке KNS-2022 в программу на языке ассемблера. Для указания выходных файлов используются входные параметры транслятора, которые описаны в пункте 2.2. 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 KNS-2022 и формирование таблицы лексем и таблицы идентификаторов.

Синтаксический анализ – это основная часть транслятора, предназначенная для распознавания синтаксических конструкций и формирования промежуточного кода KNS-2022. Для этого используются таблица лексем и идентификаторов. Синтаксический анализатор распознаёт синтаксические конструкции, выявляет синтаксические ошибки при их наличии и формирует дерево разбора

Семантический анализ в свою очередь является проверкой исходной программы KNS-2022 на семантическую согласованность с определением языка, т.е. проверяет правильность текста исходной программы с точки зрения семантики.

Генератор кода – этап транслятора, выполняющий генерацию ассемблерного кода на основе полученных данных на предыдущих этапах трансляции. Генератор кода принимает на вход таблицы идентификаторов и лексем и транслирует код на языке KNS-2022, прошедший успешно все предыдущие этапы, в код на языке Ассемблера.

## **2.2. Перечень входных параметров транслятора**

Входные параметры представлены в таблице 2.1.

Таблица 2.1 – Входные параметры транслятора языка KNS-2022

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<имя\_файла> | Входной файл с любым расширением, в котором содержится исходный код на языке KNS-2022. Данный параметр должен быть указан обязательно. В случае если он не будет задан, то выполнение этапа трансляции не начнётся. | Не предусмотрено |
| -log:<имя\_файла> | Файл содержит в себе краткую информацию об исходном коде на языке KNS-2022. | <имя\_файла>.log |
| -out:<имя\_файла> | В этот файл будет записан результат обработки исходного кода | <имя\_файла>.out |

## **2.3. Перечень протоколов, формируемых транслятором**

Результатом работы транслятора языка KNS-2022 является исходный код на языке ассемблера и протокол работы транслятора, содержащий основную информацию о процессе обработки исходного кода.

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

Таблица 2.2 – Протоколы, формируемые транслятором языка KNS-2022

|  |  |
| --- | --- |
| Формируемый протокол | Описание протокола |
| Файл журнала, “log.txt” | Файл содержит в себе краткую информацию об исходном коде на языке KNS-2022. В этот файл могут быть выведены таблицы идентификаторов, лексем, а также сообщения о возникших ошибках. |
| “KNS.asm” | Содержит сгенерированный код на языке Ассемблера. |

# **Глава 3. Разработка лексического анализатора**

## **3.1. Структура лексического анализатора**

Первая стадия работы компилятора называется лексическим анализом, а программа, её реализующая, – лексическим анализатором (сканером). На вход лексического анализатора подаётся исходный код входного языка. Лексический анализатор выделяет в этой последовательности простейшие конструкции языка. Лексический анализатор производит предварительный разбор текста, преобразующий единый массив текстовых символов в массив токенов.

Примеры лексических единиц: идентификаторы, числа, символы операций, служебные слова и т. д. Лексический анализатор преобразует исходный текст, заменяя лексические единицы их внутренним представлением – лексемами, для создания промежуточного представления исходной программы. Каждой лексеме сопоставляется ее тип и запись в таблице идентификаторов, в которой хранится дополнительная информация.

Cтруктура лексического анализатора представлена на рисунке 3.1.



Рисунок 3.1 — Структура лексического анализатора KNS-2022

## **3.2. Контроль входных символов**

Исходный код на языке программирования KNS-2022 прежде, чем транслироваться, проверяется на допустимость символов. То есть изначально из входного файла считывается по одному символу и проверяется является ли он разрешённым.

Изображение выглядит как стол

Автоматически созданное описаниеТаблица для контроля входных символов представлена на рисункеИзображение выглядит как стол

Автоматически созданное описание 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 ASCII.

Описание значения символов: T – разрешённый символ, F – запрещённый символ, A – символ-сепаратор, I – игнорируемый символ.

## **3.3. Удаление избыточных символов**

Избыточными символами являются символы табуляции и пробелы.

Избыточные символы удаляются на этапе разбиения исходного кода на лексемы.

Описание алгоритма удаления избыточных символов:

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

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

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

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

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

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

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Пояснение |
| sourse | m | Главная функция. |
| = | = | Объявление переменной. |
| function | f | Объявление функции. |
| print | p | Ввод данных. |
| return | r | Возврат функцией значения |
| while | w | Оператор цикла |
| unint, str, boolean | t | Названия типов данных языка. |
| идентификатор | i | Идентификатор |
| литерал | l | Литерал любого доступного типа. |
| expression | I | Оператор условного блока |
| $ | $ | Указание принадлежности переменной к какой-либо функции |
| ; | ; | Разделение выражений. |
| , | , | Разделение параметров функций. |
| +  -  \*  % | +  -  \*  % | Знаки арифметических операций. |
| >  <  >=  <=  &  ! | >  <  b  m  &  ! | Знаки операторов сравнения |
| { | { | Начало блока/тела функции. |
| } | } | Закрытие блока/тела функции. |

Каждому выражению соответствует детерминированный конечный автомат, по которому происходит разбор данного выражения. На каждый автомат в массиве подаётся токен и с помощью регулярного выражения, соответствующего данному графу переходов, происходит разбор. В случае успешного разбора выражения оно записывается в таблицу лексем. Если выражение является идентификатором или литералом, информация также заносится в таблицу идентификаторов.

Также в приложении [В](#_Приложение_В) находятся конечные автоматы, соответствующие лексемам языка KNS-2022.

## **3.5. Основные структуры данных**

Основные структуры таблиц лексем и идентификаторов данных языка KNS-2022, используемых для хранения, представлены в приложении А. В таблице лексем содержится лексема, её номер, полученный при разборе, номер строки в исходном коде и индекс таблицы идентификаторов. В таблице идентификаторов содержится имя идентификатора, номер в таблице лексем, тип данных, тип идентификатора и его значение.

## **3.6. Принцип обработки ошибок**

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

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

Перечень сообщений лексического анализатора представлен на рисунке 3.3.

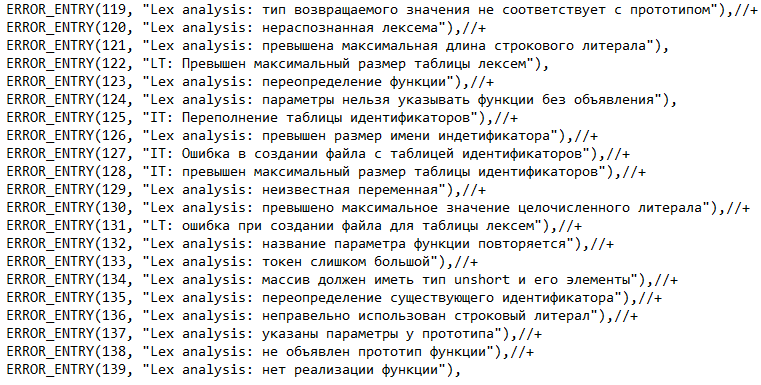


Рисунок 3.3 – Перечень ошибок лексического анализатора

## **3.8. Параметры лексического анализатора**

Входным параметром лексического анализатора является структура IN, которая содержит исходный текст программы, написанный на языке KNS-2022, а также структура LOG, которая содержит файл протокола

## **3.9. Алгоритм лексического анализа**

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

Пример. Регулярное выражение для ключевого слова false.

Граф конечного автомата для этой лексемы представлен на рисунке 3.4. S0 – начальное состояние, S5 – конечное состояние автомата. В виде кода представлен на рисунке 3.5.

e

s

l

a

f

Рисунок 3.4 – Граф переходов для цепочки ‘false’

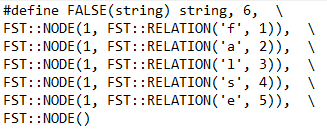


Рисунок 3.5 — Граф переходов для цепочки “ false ”

## **3.10. Контрольный пример**

Результат работы лексического анализатора – таблицы лексем и идентификаторов – представлен в приложении [Б](#_Приложение_Б).

# **Глава 4. Разработка синтаксического анализатора**

## **4.1. Структура синтаксического анализатора**

Синтаксический анализ – это фаза трансляции, выполняемая после лексического анализа и предназначенная для распознавания синтаксических конструкций KNS-2022. 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 KNS-2022

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

В синтаксическом анализаторе транслятора языка KNS-2022 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 KNS-2022 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 KNS-2022

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| S | tfi(F){NrE;}S  d{N}S  s{NrB;}  tfi(){NrE;}S | Порождает правила, описывающее общую структуру программы |
| N | i(W);N  i(W);  ti;N  ti;  t$i$i;N  t$i$i;  tfi(F);N  tfi();N  tfi(F);  tfi();  t[l]iL;N  t[l]iL;  t$i$ [l]iL;N  t$i$ [l]iL;  ieE;N  ieE;  pB;N  pB;  I(C){N}N  I(C){N}  I(C){rE;}N  I(C){rE;}  I(C){NrE;}N  I(C){NrE;}  w(C){N}N  w(C){N} | Порождает правила, описывающие конструкции языка |
| E | i  l  i[i]  i[l]  i[i]M  i[l]M  (E)  i(W)  i(W)M  iM  lM  (E)M | Порождает правила, описывающие выражения |
| L | l  l, L | Порождает правила, описывающие инициализацию массива |
| B | i  l  i[i]  i[l] | Порождает правила, описывающие операнды логического выражения |
| M | +E  -E  \*E  %E  +EM  -EM  \*EM  %EM | Порождает правила, описывающие арифметические операции |
| C | iOB  lOB | Порождает правила, описывающие строение логической инструкции |
| O | <  >  !  b  &  m | Порождает правила, описывающие логические операции |
| F | ti  ti, F | Порождает правила, описывающие параметры функций |
| W | i  l  i, W  l, W | Порождает правила, определяющие выражения, которые могут быть переданы в функции в качестве параметров |

Продолжение таблицы 4.1

## **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 [Г](#_Приложение_Г).

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 А) |
|  | 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 |

Продолжение таблицы 4.2

## **4.4. Основные структуры данных**

Основные структуры данных синтаксического анализатора включают в себя структуру магазинного конечного автомата и структуру грамматики Грейбах, описывающей правила языка KNS-2022. Данные структуры представлены в приложении [Г](#_Приложение_Г).

## **4.5. Описание алгоритма синтаксического разбора**

Принцип работы автомата следующий:

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

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

Перечень сообщений синтаксического анализатора представлен на рисунке 4.2.

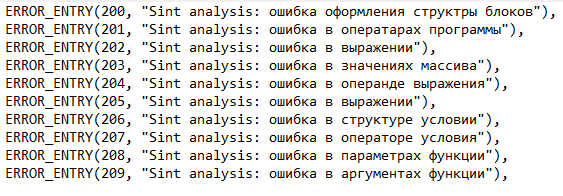


Рисунок 4.2 – Перечень сообщений синтаксического анализатора

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

Входным параметром синтаксического анализатора является структура LexTable, которая содержит сформированную таблицу лексем, полученную на этапе лексического анализа, потоки вывода протокола, а также правила контекстно-свободной грамматики в форме Грейбах.

Выходными параметрами являются трассировка прохода таблицы лексем и правила разбора, которые записываются в файл протокола.

## **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 в протокол будет выведено диагностическое сообщение.

## **4.9. Контрольный пример**

Пример разбора синтаксическим анализатором исходного кода на языке KNS-2022 представлен в приложении [Е](#_Приложение_Е).

# **Глава 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). Структура семантического анализатора представлена на рисунке 5.1.



Рисунок 5.1 — структура семантического анализатор

## **5.2. Функции семантического анализатора**

Назначение семантического анализа – проверка смысловой правильности

конструкций языка программирования

Входные данные для семантического анализатора:

* таблица идентификаторов;
* дерево разбора – результат разбора синтаксических конструкций входного языка.

**Основные действия семантического анализатора:**

1) проверка соблюдения в исходной программе семантических правил

входного языка;

2) дополнение внутреннего представления программы в компиляторе

oпeраторами и действиями, неявно предусмотренными семантикой

входного языка;

3) проверка элементарных семантических (смысловых) норм языка

программирования.

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

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

Сообщения, формируемые семантическим анализатором, представлены на рисунке 5.2.

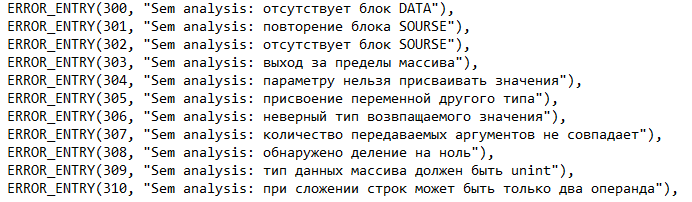


Рисунок 5.2 – Перечень сообщений семантического анализатора

## **5.4. Принцип обработки ошибок**

Семантический анализатор - проверяет, что объявления и утверждения программы семантически верны. Например: соответствие типов данных в выражении, совпадение фактических и формальных параметров функции.

Обработка ошибок происходит следующим образом:

1. Анализатор перебирает таблицу лексем.
2. Проверяет исключительные ситуации, которые могут быть связаны с данной лексемой.
3. В случае нахождения ошибки, она записывается в log журнал и дублируется в консоль

## **5.5. Контрольный пример**

Результат работы является успешным только тогда, когда программа после проверки семантическим анализом не обнаружила никаких ошибок.

# **Глава 6. Вычисление выражений**

## **6.1. Выражения, допускаемые языком**

В языке KNS-2022 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 KNS-2022

|  |  |
| --- | --- |
| Операция | Значение приоритета |
| ( ) | 3 |
| \* | 2 |
| % | 2 |
| + | 1 |
| - | 1 |
| } | 0 |
| { | 0 |

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

Выражения в языке KNS-2022 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 выражений, в которой операторы расположены после своих операндов. Выражение в обратной польской нотации читается слева направо: операция выполняется над двумя операндами, непосредственно стоящими перед знаком этой операции.

Алгоритм построения:

– читаем очередной символ;

– если он является идентификатором или литералом, то добавляем его к выходной строке;

– если символ является символом функции, то помещаем его в стек;

– если символ является открывающей скобкой, то она помещается в стек;

– исходная строка просматривается слева направо;

– если символ является закрывающей скобкой, то выталкиваем из стека в выходную строку все символы пока не встретим открывающую скобку. При этом обе скобки удаляются и не попадают в выходную строку;

– как только входная лента закончится все символы из стека выталкиваются в выходную строку;

– в случае если встречаются операции, то выталкиваем из стека в выходную строку все операции, которые имеют выше приоритетность чем последняя операция;

– также, если идентификатор является именем функции, то он заменяется на спецсимвол «@».

Таблица 6.2 – Пример преобразования выражения в обратную польскую запись

|  |  |  |
| --- | --- | --- |
| Исходная строка | Результирующая строка | Стек |
| i\*i-(i(i,i)-(i-l))%l |  |  |
| \*i-(i(i,i)-(i-l))%l | i |  |
| i-(i(i,i)-(i-l))%l | i | \* |
| -(i(i,i)-(i-l))%l | ii | \* |
| (i(i,i)-(i-l))%l | ii\* | - |
| i(i,i)-(i-l))%l | ii\* | -( |
| (i,i)-(i-l))%l | ii\*i | -(@ |
| i,i)-(i-l))%l | ii\*i | -(@( |
| ,i)-(i-l))%l | ii\*ii | -(@( |
| i)-(i-l))%l | ii\*ii | -(@( |
| )-(i-l))%l | ii\*iii | -(@( |
| -(i-l))%l | ii\*iii@2 | -( |
| (i-l))%l | ii\*iii@2 | -(- |
| i-l))%l | ii\*iii@2 | -(-( |
| -l))%l | ii\*iii@2i | -(-( |
| l))%l | ii\*iii@2i | -(-(- |
| ))%l | ii\*iii@2il | -(-(- |
| )%l | ii\*iii@2il- | -(- |
| %l | ii\*iii@2il-- | - |
| l | ii\*iii@2il-- | -% |
|  | ii\*iii@2il--l | -% |
|  | ii\*iii@2il--l% | - |
|  | ii\*iii@2il--l%- |  |

## **6.3. Программная реализация обработки выражений**

Программная реализация алгоритма преобразования выражений в обратный польский формат основана функции PolishNotation. Функция принимает как параметр адрес таблицы лексем и содержит цикл, в ходе которого перебираются все лексемы исходного кода. Если последовательность лексем соответствует началу выражения, проводится точечное преобразование выражений к польской нотации.

## **6.4. Контрольный пример**

Пример преобразования выражений из контрольных примеров к обратной польской записи представлен в таблице 6.2. Преобразование выражений в формат польской записи необходимо для построения более простых алгоритмов их вычисления и преобразования к ассемблерному коду. В приложении [И](#_Приложение_И) приведены изменённая таблицы лексем, отображающая результаты преобразования выражений в польский формат.

# **Глава 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.

Структура генератора кода представлена на рисунке 7.1.



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 KNS-2022 размещены в сегменте данных(.data). Литералы – в сегменте констант (.const). Соответствия между типами данных идентификаторов на языке KNS-2022 и на языке ассемблера приведены в таблице 7.1.

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

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке KNS-2022 | Тип идентификатора на языке ассемблера | Пояснение |
| unint | SDWORD | Хранит целочисленный тип данных без знака. |
| str | BYTE | Хранит указатель на начало строки. |
| boolean | SDWORD | Хранит целочисленный 0 или 1. |
| literal | BYTE  SDWORD | Литералы: символьные,  целочисленные |

## **7.3. Статическая библиотека**

В языке KNS-2022 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++.

Функции из стандартной библиотеки содержатся в проекте StaticLibrary, в свойствах которого указан тип конфигурации «статическая библиотека». Подключение библиотеки происходит с помощью includelib на этапе генерации кода.

Объявление функций статической библиотеки генерируется автоматически в коде ассемблера. Объявление функций статической библиотеки генерируется автоматически.

Все функции статической библиотеки представлены в пункте 1.17.

## **Особенности алгоритма генерации кода**

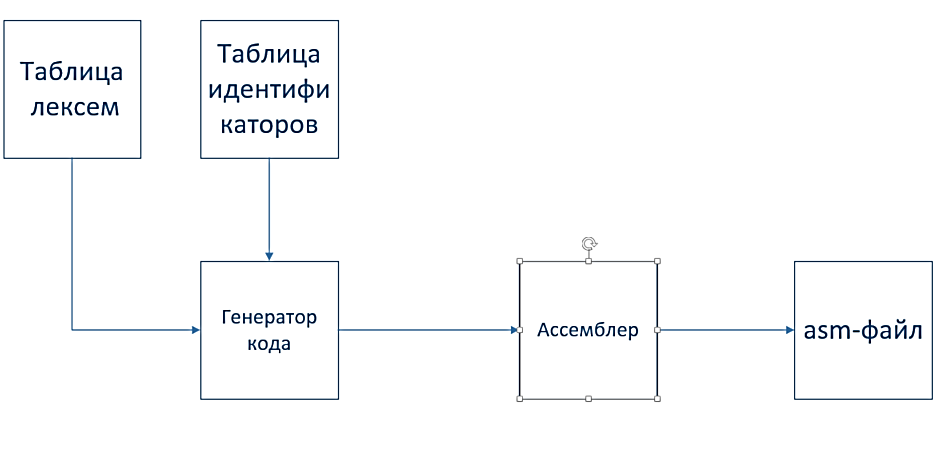
В языке KNS-2022 генерация кода строится на основе таблиц лексем и идентификаторов. Общая схема работы генератора кода представлена на рисунке 7.2.

Рисунок 7.2 – Структура генератора кода

Генерация кода языка KNS-2022 происходит поэтапно. Сразу записывается типичные данные (модель памяти, соглашение о вызовах),потом прототипы функций статической библиотеки для языка программирования KNS-2022.В блок ‘const’ записываются все литералы, а после в блок ‘data’ записываются все переменные. После этого записываются блоки всех функций, все действия внутри этого блока транслируются в код ассемблера с помощью конструкций switch и case. В завершении в файл с расширением записывается конец главной процедуры ‘sourse’ и завершение программы.

## **7.5. Входные параметры, управляющие генерацией кода**

На вход генератору кода поступают таблицы лексем и идентификаторов исходного код программы на языке KNS-2022. Результаты работы генератора кода выводятся в файл с расширением .asm.

## **7.6. Контрольный пример**

Результат генерации ассемблерного кода на основе контрольного примера из приложения А приведен в приложении [З](#_Приложение_З).

# **Глава 8. Тестирование транслятора**

## **8.1. Общие положения**

В языке KNS-2022 не разрешается использовать запрещённые входным алфавитом символы. Допустимость входных символов проверяется при считывании данных из файла, если символ запрещён, то компилятор генерирует исключение.

При этапе лексического анализа, компилятор пытается распознать входящее слово. Если слово не распознано, то компилятор генерирует исключение.

Во время проверки входных данных синтаксическим анализатором, проверяется порядок использования лексем, то есть их структура. Если подходящее правило не найдено, то синтаксический анализатор диагностирует ошибку и генерирует исключение.

Так же в языке программирования KNS-2022 используется семантический анализатор. Он в свою очередь проверяет входящие данные на семантику. При неправильной семантике он также генерирует исключение.

## **Результаты тестирования**

Тестирование входных символов на допустимость, представлено в таблице 8.1.

Таблица 8.1 – Тестирование фазы проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| dataи | Ошибка 111: недопустимый символ в исходном файле (-in), строка 1 символ 5 |

Результаты тестирования лексического анализатора показаны в таблице 8.2.

Таблица 8.2 – Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| data  {  unint x;  unint y;  unint y;  … | Ошибка 135: Lex analysis: переопределение существующего идентификатора, строка 5 символ 8 |
| data  {  unint x;  unint y123;  … | Ошибка 120: Lex analysis: нераспознанная лексема, строка 4 символ 11 |

Результаты тестирования синтаксического анализатора показаны в таблице 8.3.

Таблица 8.3 – Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| data  {  unint x;;  unint y;  … | Ошибка 201: Sint analysis: ошибка в оператарах программы , строка 2 |
| data  {  …  unint function sum(unint a,, unint b); | Ошибка 208: "Sint analysis: ошибка в параметрах функции , строка 13 |

Итоги тестирования семантического анализатора приведены в таблице 8.4.

Таблица 8.4 – Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| unint function sum(unint a, unint b)  {  …  }  …  z = x+x\*(x\*y+sum() - x);  … | Ошибка 307: Sem analysis: количество передаваемых аргументов не совпадает |
| …  int function sum(int a, int b)  {  a equals 12;  … | Ошибка 304: Sem analysis: параметру нельзя присваивать значения |

# **Заключение**

В результате курсовой работы был разработан транслятор, который состоит из лексического, синтаксического и семантического анализаторов, а так же и генератор кода для языка программирования KNS-2022. По завершению курсового проекта, все поставленные задачи были выполнены успешно:

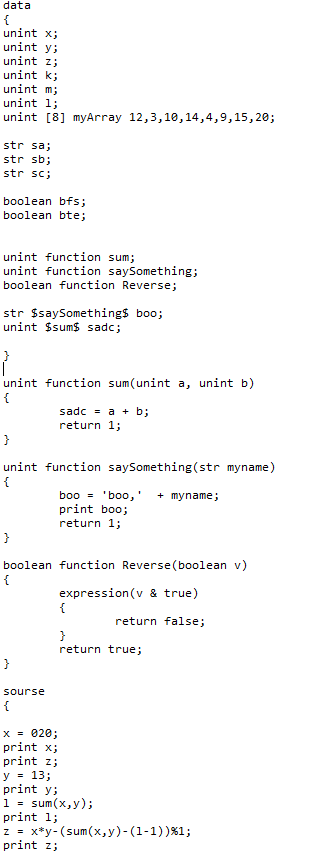
1. Сформулирована спецификация языка KNS-2022;
2. Разработаны конечные автоматы и важные алгоритмы на их основе для эффективной работы лексического анализатора;
3. Осуществлена программная реализация лексического анализатора, распознающего допустимые цепочки спроектированного языка;
4. Разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
5. Осуществлена программная реализация синтаксического анализатора;
6. Разработан семантический анализатор, осуществляющий проверку используемых инструкций на соответствие логическим правилам;
7. Разработан транслятор кода на язык ассемблера;
8. Проведено тестирование всех вышеперечисленных компонентов.

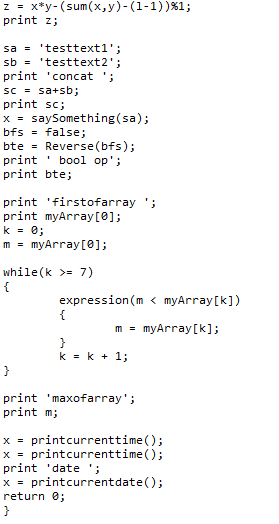
Выполнение курсового проекта, позволило укрепить свои знания, а также узнать более подробно в разных аспектах языках C++ и ассемблера. Так же это закрепило понимание механизма обработки различных языков программирования.

# **Приложения**

## **Приложение А**

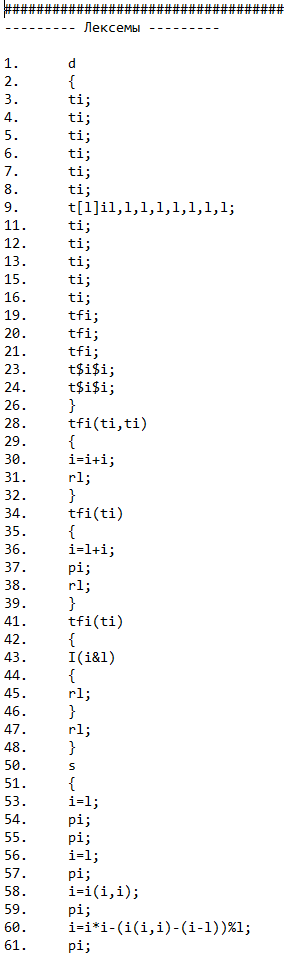
Контрольный пример:





## **Приложение Б**

Таблица лексем:



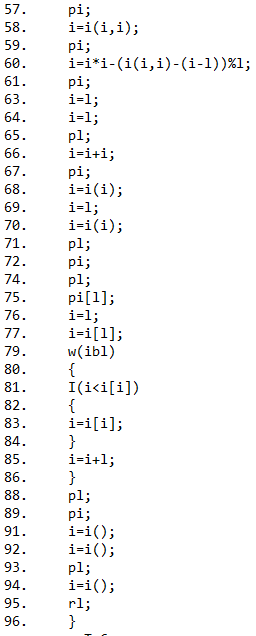
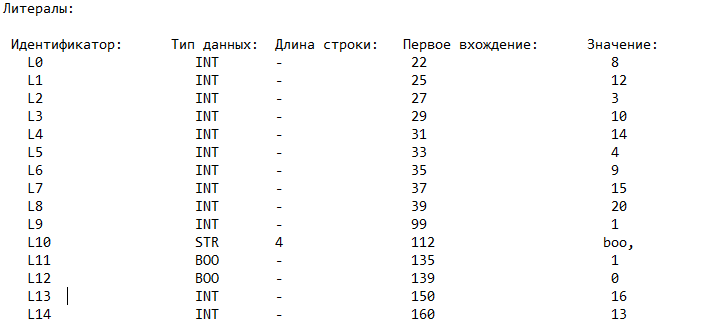
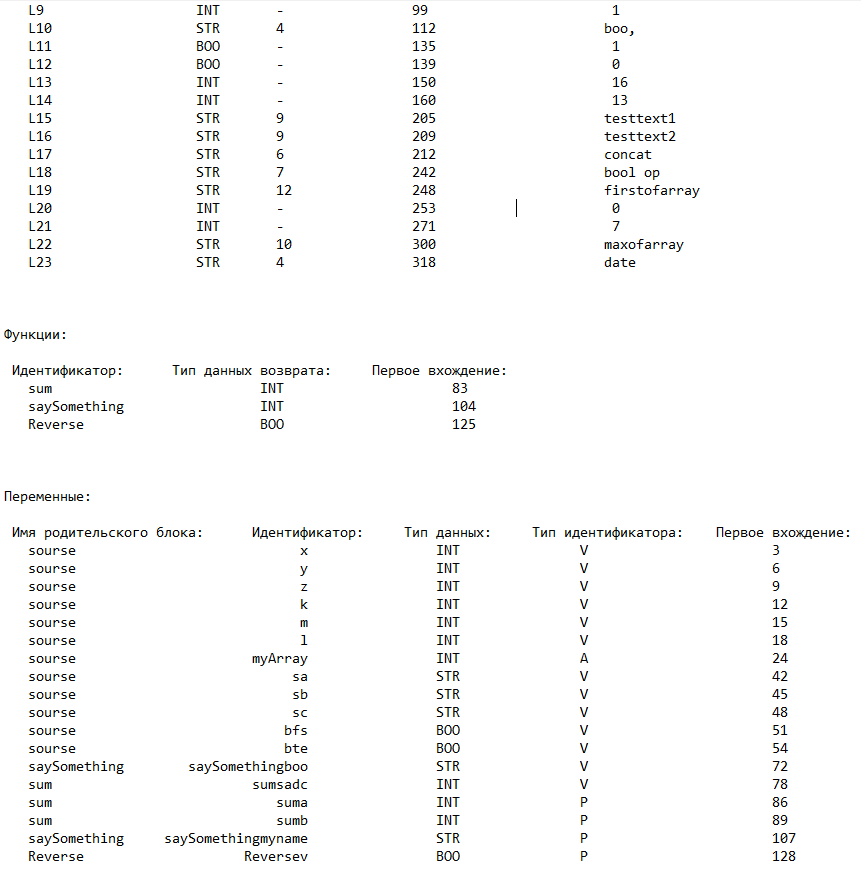
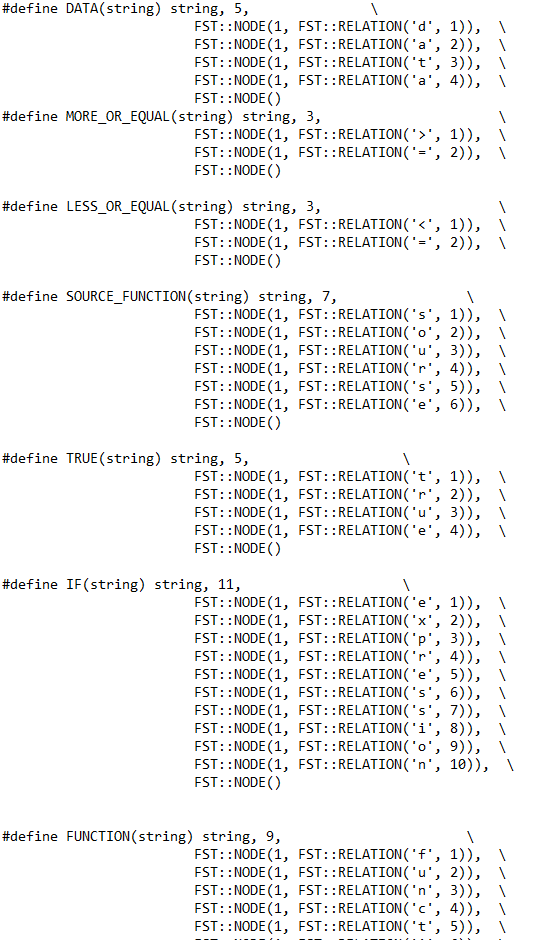
****

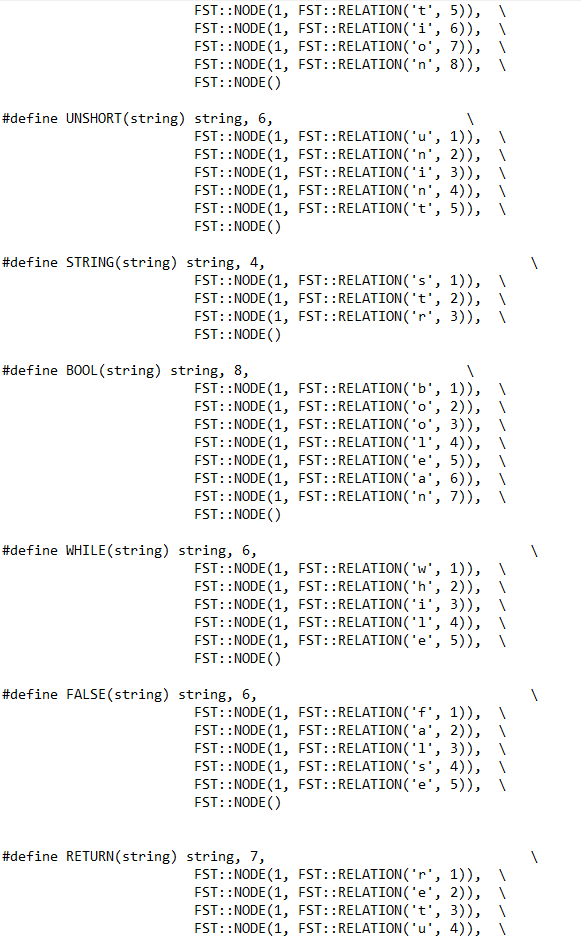
Таблица идентификаторов:

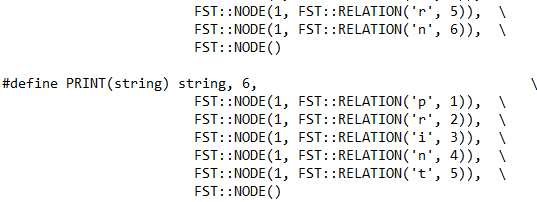


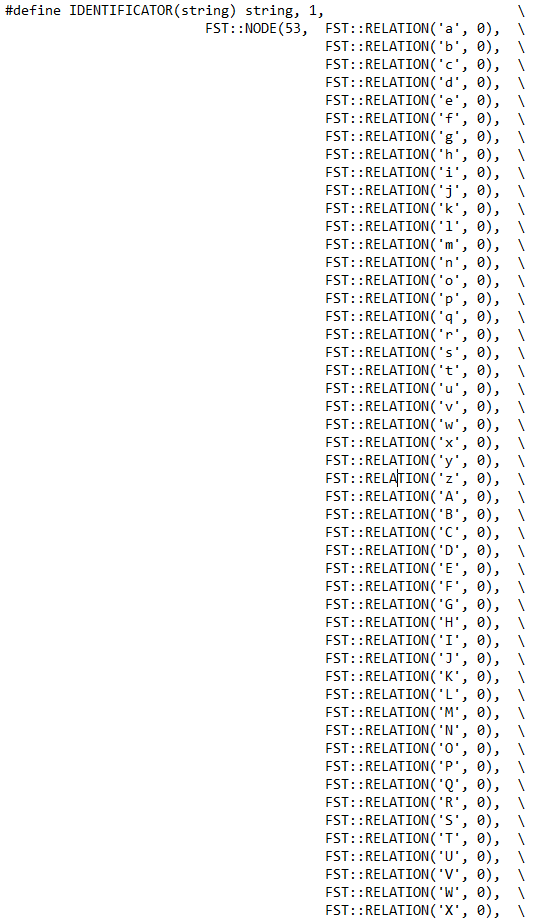


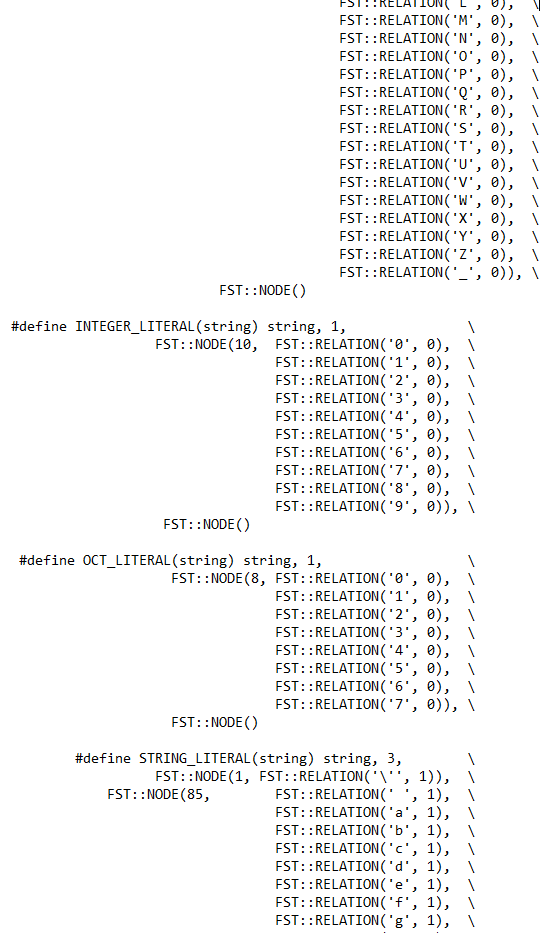
## **Приложение В**

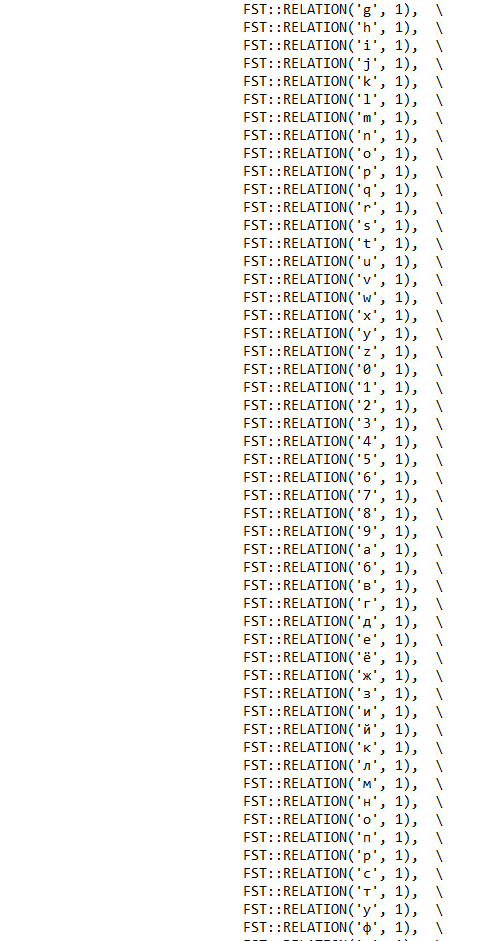


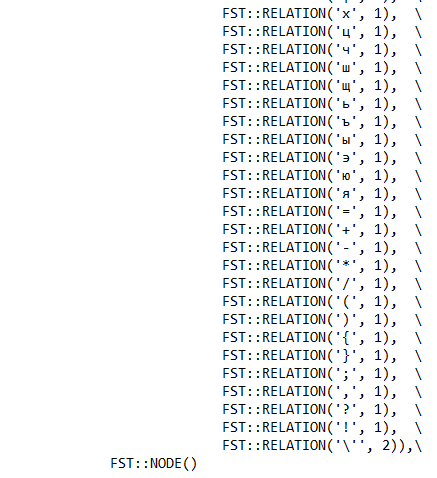






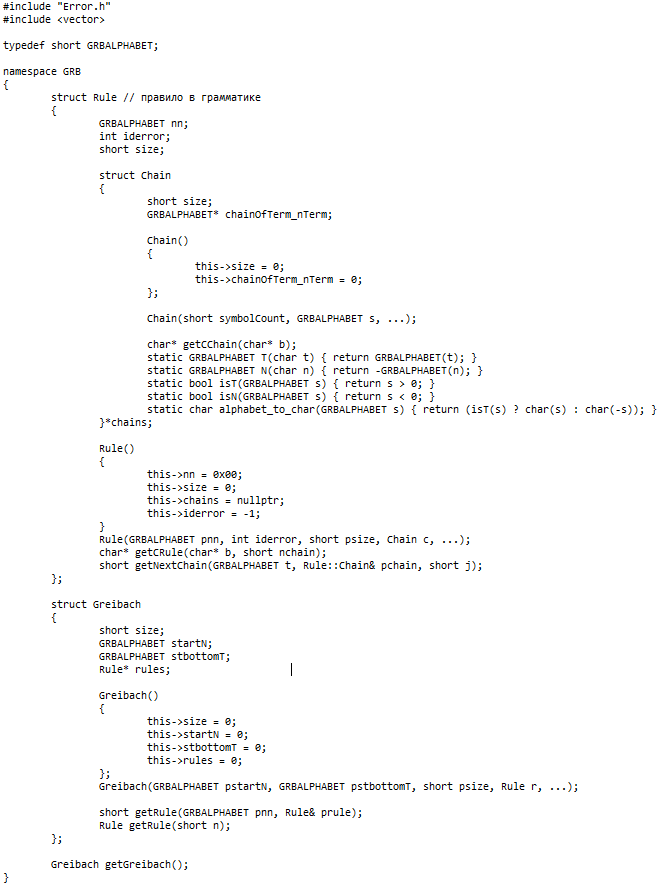




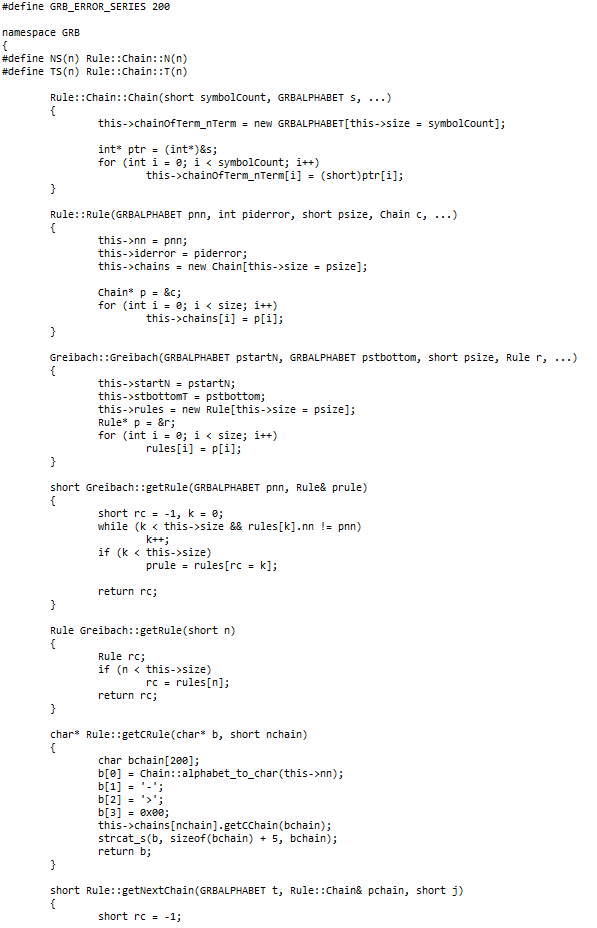


## **Приложение Г**

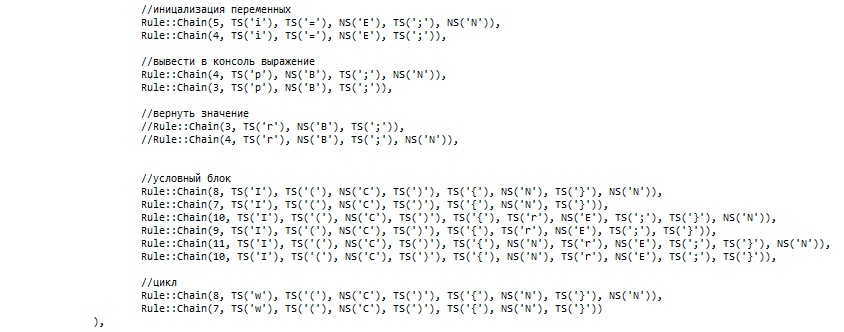
Файл GRB.h:

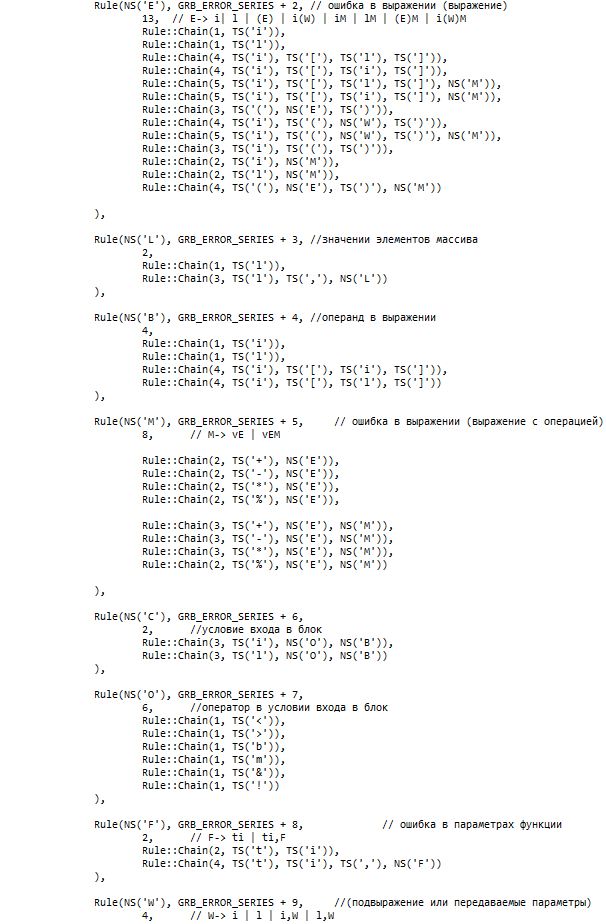


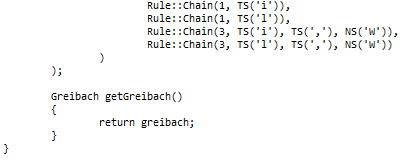
Файл GRB.cpp:











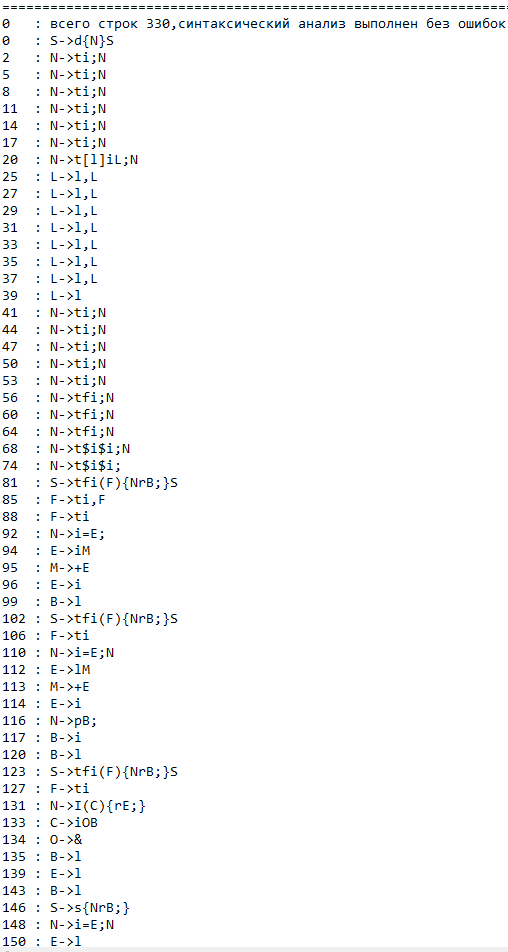
## **Приложение Д**

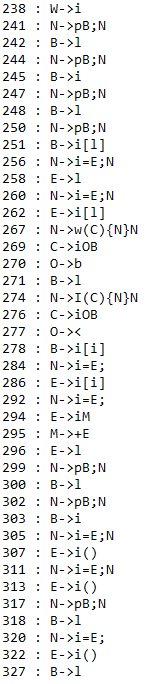
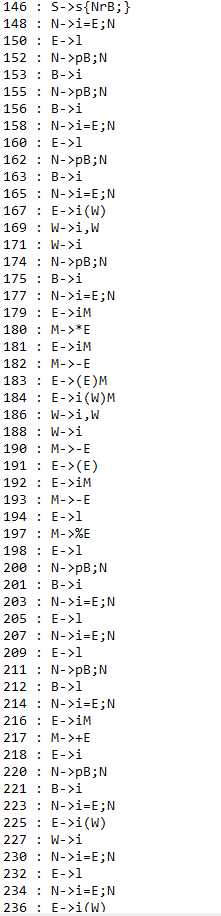
Файл MFST.h: 



## **Приложение Е**

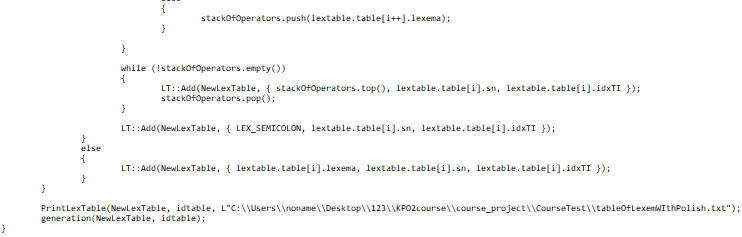
Разбор синтаксическим анализатором:



****

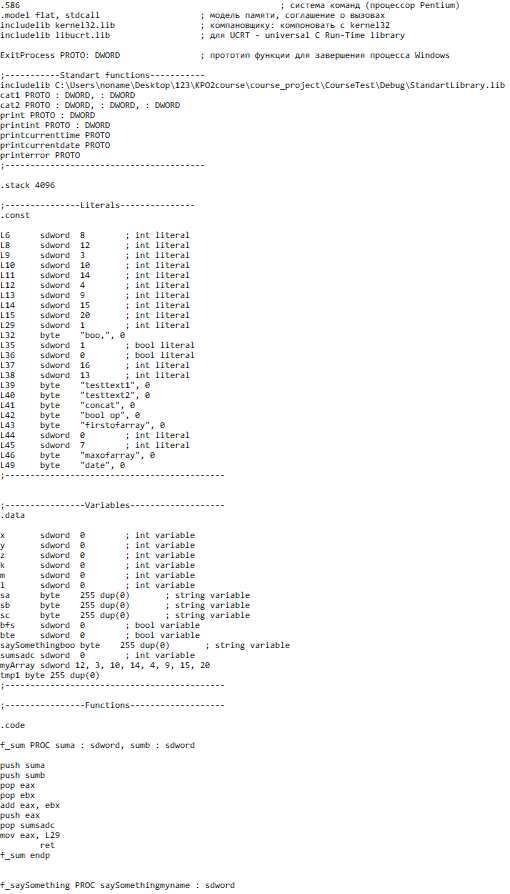
## **Приложение Ж**

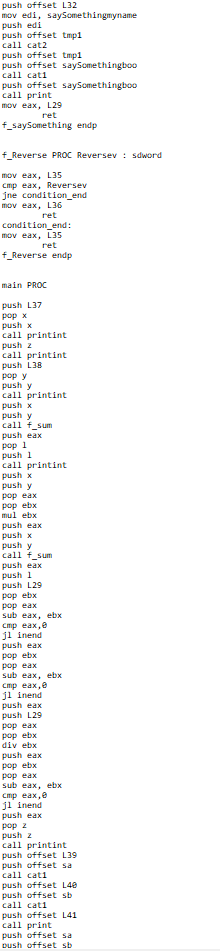
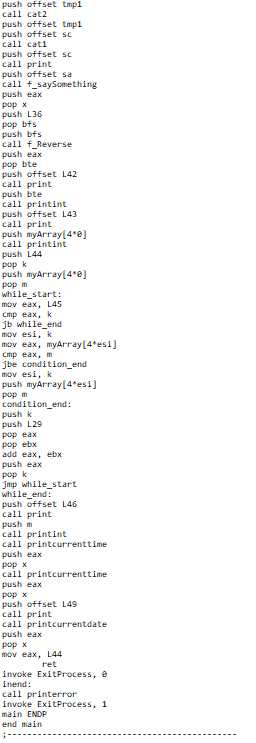
Файл PolishNotation.cpp:

****

## **Приложение З**

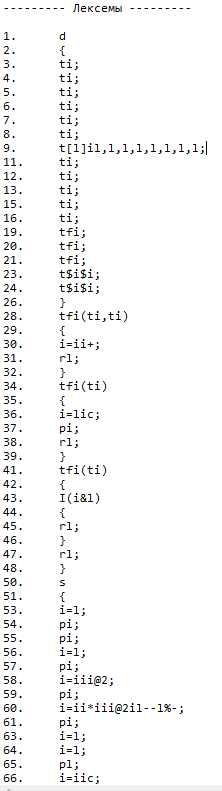
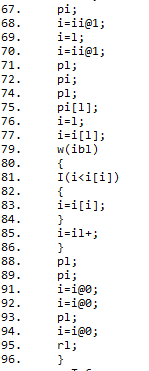
Файл KNS.asm :





## **Приложение И**

Результат преобразования выражений:



# **Список использованной литературы**

1. Ахо А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003. – 768с.
2. Ахо, А. Теория синтаксического анализа, перевода и компиляции /А. Ахо, Дж. Ульман. – Москва : Мир, 1998. – Т. 2 : Компиляция. - 487 с.
3. Ирвин К. Р. Язык ассемблера для процессоров Intel / К. Р. Ирвин. – M.: Вильямс, 2005. – 912с.
4. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. - 3-е изд. – Москва : Вильямс, 2003. - 429 с.