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 KKV-2020»

Выполнил студент Квит Кирилл Витальевич

(Ф.И.О.)

Руководитель проекта cт. пр. Пахолко Алёна Степановна

(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. Пахолко Алёна Степановна

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

Нормконтролер cт. пр. Пахолко Алёна Степановна

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

Курсовой проект защищен с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Минск 2020

**Содержание**

[**Глава 1. Спецификация языка программирования** 5](#_Toc58582027)

[**1.1 Характеристика языка программирования** 5](#_Toc58582028)

[**1.2 Алфавит языка** 5](#_Toc58582029)

[**1.3 Применяемые кодировки** 5](#_Toc58582033)

[**1.4 Типы данных** 6](#_Toc58582034)

[**1.5 Преобразование типов данных** 6](#_Toc58582035)

[**1.6 Идентификаторы** 6](#_Toc58582036)

[**1.7 Литералы** 6](#_Toc58582037)

[**1.8 Объявление данных** 7](#_Toc58582038)

[**1.9 Инициализация данных** 7](#_Toc58582039)

[**1.10 Инструкции языка** 7](#_Toc58582040)

[**1.11 Операции языка** 7](#_Toc58582041)

[**1.12 Выражения и их вычисления** 8](#_Toc58582042)

[**1.13 Программные конструкции языка** 8](#_Toc58582043)

[**1.14 Область видимости** 8](#_Toc58582044)

[**1.15 Семантические проверки** 8](#_Toc58582045)

[**1.16 Распределение оперативной памяти на этапе выполнения** 9](#_Toc58582046)

[**1.17 тандартная библиотека и её состав** 9](#_Toc58582047)

[**1.18 Ввод и вывод данных** 10](#_Toc58582049)

[**1.19 Точка входа** 10](#_Toc58582050)

[**1.20 Препроцессор** 10](#_Toc58582051)

[**1.21 Соглашения о вызовах** 10](#_Toc58582052)

[**1.22 Классификация сообщений транслятора** 10](#_Toc58582053)

[**1.23 Контрольный пример** 11](#_Toc58582055)

[**Глава 2. Структура транслятора** 12](#_Toc58582056)

[**2.1 Компоненты транслятора, их назначение и принципы взаимодействия** 12](#_Toc58582057)

[**2.2 Перечень входных параметров транслятора** 13](#_Toc58582058)

[**2.3 Перечень протоколов, формируемых транслятором и их содержимое** 13](#_Toc58582060)

[**3.1 Структура лексического анализатора** 14](#_Toc58582061)

[**3.2 Контроль входных символов** 15](#_Toc58582062)

[**3.3 Удаление избыточных символов** 15](#_Toc58582063)

[**3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов** 15](#_Toc58582064)

[**3.5 Основные структуры данных** 16](#_Toc58582065)

[**3.6 Принцип обработки ошибок** 16](#_Toc58582066)

[**3.7 Структура и перечень сообщений лексического анализатора** 17](#_Toc58582067)

[**3.8 Параметры лексического анализатора и режимы его работы** 17](#_Toc58582068)

[**3.9 Алгоритм лексического анализа** 17](#_Toc58582069)

[**3.10 Контрольный пример** 17](#_Toc58582070)

[**Глава 4. Разработка синтаксического анализатора** 18](#_Toc58582071)

[**4.1 Структура синтаксического анализатора** 18](#_Toc58582072)

[**4.2 Контекстно-свободная грамматика, описывающая синтаксис языка** 18](#_Toc58582073)

[**4.3 Построение конечного магазинного автомата** 19](#_Toc58582075)

[**4.4 Основные структуры данных** 20](#_Toc58582076)

[**4.5 Описание алгоритма синтаксического разбора** 20](#_Toc58582077)

[**4.6 Структура и перечень сообщений синтаксического анализатора** 20](#_Toc58582078)

[**4.7 Параметры синтаксического анализатора и режимы его работы** 20](#_Toc58582079)

[**4.8 Принцип обработки ошибок** 20](#_Toc58582080)

[**4.9 Контрольный пример** 21](#_Toc58582081)

[**Глава 5. Разработка семантического анализатора** 22](#_Toc58582082)

[**5.1 Структура семантического анализатора** 22](#_Toc58582083)

[**5.2 Функции семантического анализатора** 22](#_Toc58582084)

[**5.3 Структура и перечень сообщений семантического анализатора** 22](#_Toc58582085)

[**5.4 Принцип обработки ошибок** 22](#_Toc58582086)

[**5.5 Контрольный пример** 22](#_Toc58582087)

[**Глава 6. Преобразование выражений** 23](#_Toc58582088)

[**6.1 Выражения, допускаемые языком** 23](#_Toc58582089)

[**6.2 Польская запись и принцип её построения** 23](#_Toc58582090)

[**6.4 Контрольный пример** 24](#_Toc58582091)

[**Глава 7. Генерация кода** 25](#_Toc58582092)

[**7.1 Структура генератора кода** 25](#_Toc58582093)

[**7.2 Представление типов данных в оперативной памяти** 25](#_Toc58582094)

[**7.3 Алгоритм работы генератора кода** 25](#_Toc58582095)

[**7.4 Состав статической библиотеки** 26](#_Toc58582096)

[**7.5 Контрольный пример** 26](#_Toc58582097)

[**Глава 8. Тестирование транслятора** 27](#_Toc58582098)

[**8.1 Тестирование фазы проверки на допустимость символов** 27](#_Toc58582099)

[**8.2 Тестирование лексического анализатора** 27](#_Toc58582101)

[**8.3 Тестирование синтаксического анализатора** 27](#_Toc58582103)

[**8.4 Тестирование семантического анализатора** 27](#_Toc58582105)

[**Заключение** 45](#_Toc58582106)

[**Приложение** **А** 29](#_Toc58582107)

[**Приложение Б** 33](#_Toc58582108)

[**Приложение В** 38](#_Toc58582109)

[**Приложение Г** 39](#_Toc58582110)

[**Приложение Д** 41](#_Toc58582111)

[**Литература** 45](#_Toc58582112)

**Введение**

Целью курсового проекта поставлена задача разработки компилятора для моего языка программирования KKV-2020. Этот язык программирования предназначен для выполнения простейших операций и арифметических действий над числами.

Компилятор KKV-2020 – это программа, задачей которого является перевод программы, написанной на языке программирования KKV-2020 в программу на язык ассемблера.

Транслятор KKV-2020 состоит из следующих частей:

– семантический анализатор;

– синтаксический анализатор;

– логический анализатор;

– генератор исходного кода на языке ассемблера.

Исходя из цели курсового проекта, были определены следующие задачи:

– разбработка спецификации языка программирования;

– разбратка структуры транслятора;

– разработка лексического и семантического анализаторов;

– разработка синтаксического анализатора;

– преобразование выражений;

– генерация кода на язык ассемблера;

– тестирование транслятора.

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

# **Глава 1. Спецификация языка программирования**

## **Характеристика языка программирования**

Язык KKV-2020 – это процедурный, универсальный, строго типизированный, компилируемый язык. Не является объектно-ориентированным.

* 1. **Алфавит языка**

Таблица используемых символов представлена на рисунке 1.1

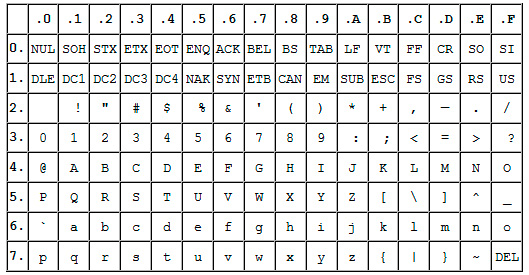


Рисунок 1.1 – Алфавит входных символов

**Символы сепараторы**

Символы сепараторы приведены в таблице 1.1.

Таблица 1.1 – Символы сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение |
| пробел | Допускается везде кроме идентификаторов и ключевых слов. |
| ; | Конец строки исходного кода, конец программного блока. |
| , | Разделение параметров функции. |
| () | Параметров функций, приоритетность операций в выражениях. |
| {} | Программный блок. |
| [] | Блок для цикла |

## Использование символов сепараторов представлено в Приложении Д.

## **Применяемые кодировки**

Для написания исходного кода на языке KKV-2020 используется кодировка Windows-1251.

## **Типы данных**

Описание типов данных представлено в таблице 1.2.

Таблица 1.2 – Типы данных языка KKV-2020

|  |  |
| --- | --- |
| Тип данных | Описание |
| dec (целый) | Занимаемое место в памяти: 4 байта.  Диапазон чисел: от 0 до .  Возможные операции:   * *арифметические*   + – бинарный, суммирование;  - – бинарный, разность;  \* – бинарный, умножение;  / – бинарный, деление;  % – бинарный, остаток от деления;  = – присваивание значения;  Автоматическая инициализация значением 0. |
| str (строковый) | Занимаемое место в памяти: 1 байт \* кол-во символов строки. Автоматическая инициализация nullptr. |

## **Преобразование типов данных**

Преобразование типов данных не поддерживается.

## **Идентификаторы**

Описание идентификаторов:

* Максимальная длина: 15 символов.
* Используются буквы латиницы верхнего регистра и нижнего регистров, а также цифры.
* Идентификатор не может совпадать с ключевыми словами.

## **Литералы**

Описание литералов приведено в таблице 1.3.

Таблица 1.3 – Литералы языка KKV-2020

| Литерал | Описание |
| --- | --- |
| целочисленный | Целочисленные литералы типа dec.  Представление:   * Двоичное: целые числа [0, 1]; * Диапазон значений: от 0 до . |
| строковый | Строковые литералы типа str.  Представление: заключаются в '…'(одинарные кавычки), используются буквы латиницы и кириллицы [a…z], [A…Z], [а…я], [А…Я]. |

## **Объявление данных**

Объявление переменных должно быть внутри программного блока.

Синтаксис объявления: new <тип данных> <идентификатор>; Использование переменной за границей программного блока, в которой она была объявлена, запрещено.

## **Инициализация данных**

Инициализация переменной происходит внутри программного блока, внутри которого она была объявлена. Присвоение значения переменной вне её блока видимости запрещена. Синтаксис инициализации: <идентификатор> = <значение>;

## **Инструкции языка**

Инструкции языка KKV-2020 представлены в таблице 1.4.

Таблица 1.4 – Инструкции языка KKV-2020

|  |  |
| --- | --- |
| Инструкция | Синтаксис |
| Объявление переменной | new <тип данных> <идентификатор>; |
| Присваивание значения | <идентификатор> = <литерал>; |
| Возврат значения | ret <идентификатор или литерал>; |
| Вывод данных | console <идентификатор или литерал>; |

## **Операции языка**

Операции языка KKV-2020 приведены в таблице 1.5.

Таблица 1.5 – Операции языка KKV-2020

| Операция | Приоритетность |
| --- | --- |
| ( | 0 |
| ) | 0 |
| , | 1 |
| + | 2 |
| **-** | 2 |
| **\*** | 3 |
| **/** | 3 |
| **%** | 3 |

Максимальным значением приоритетности является “3”, минимальным “0”.

## **Выражения и их вычисления**

В выражениях могут использоваться: +, -, \*, /, %, () – для изменения приоритета операций. Так же в выражениях может присутствовать вызов функции. Арифметические операции можно проводить только с целочисленными значениями. Выражением считается строка, идущая после знака ‘=’.

## **Программные конструкции языка**

Программные конструкции языка KKV-2020 приведены в таблице 1.6.

Таблица 1.6 – Программные конструкции языка KKV-2020

|  |  |
| --- | --- |
| Конструкция | Синтаксис |
| Главная функция  (точка входа) | main  {  …  ret < идентификатор или литерал >;  }; |
| Функции | <тип данных> func <идентификатор>(<тип данных> <идентификатор>, …)  {  …  ret < идентификатор или литерал >;  }; |
| Цикл | while (<dec-идентификатор>)  [  …  ] |

## **Область видимости**

Организация: «сверху вниз» (по принципу С++). Перед использованием переменной требуется её объявить. Все переменные объявляются и используются только внутри программного блока. В разных блоках допускается объявление одноименных переменных. Объявление глобальных переменных запрещено.

## **Семантические проверки**

Перечень семантических проверок приведен в таблице 1.7.

Таблица 1.7 – Семантические проверки языка KKV-2020

|  |  |
| --- | --- |
| Номер | Ошибки |
| 1 | Ошибка в возвращаемом значении |
| 2 | Ошибка в параметрах функции |
| 3 | Ошибка в параметрах функции библиотеки |
| 4 | Ошибка в параметрах функции библиотеки (неверное кол-во аргументов) |
| 5 | Несоответствие присваиваемого типа данных |
| 6 | Несоответствие типа функции и возвращаемого значения |
| 7 | Main Должен возвращать числовое значение |
| 8 | Функция Copy принимает литерал в качестве строки-назначения |
| 9 | Присвоение пустой строки |
| 10 | Операция со строками |
| 11 | В цикл передано не числовое значение |

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

Все переменные размещаются в стеке.

## **Стандартная библиотека и её состав**

Состав статической библиотеки приведен в таблице 1.8.

Таблица 1.8 – Состав статической библиотеки языка KKV-2020

| Имя функции | Возвращаемое значение | Принимаемые параметры | Описание |
| --- | --- | --- | --- |
| Concat | str  (Строковое) | Str1 – первая строка  Str2 – вторая строка | Возвращает строку состоящую из двух принимаемых |
| Copy | dec  (целочисленное) | origin – строка для копирования  destination – строка для вставки | Копирует строку origin в строку destination |

## **Ввод и вывод данных**

Ввод данных не поддерживается языком программирования KKV-2020. console <идентификатор или литерал>; – вывод в стандартный поток вывода.

## **Точка входа**

Точкой входа является функция main.

## **Препроцессор**

Предпроцессор в языке KKV-2020 не предусмотрен.

## **Соглашения о вызовах**

В языке вызов функций происходит по соглашению о вызовах stdcall. **Объектный код**KKV-2020 транслируется в язык ассемблера.

## **Классификация сообщений транслятора**

В случае возникновения ошибки в коде программы на языке KKV-2020 и вы явления её транслятором в текущий файл протокола выводится сообщение. Перечень ошибок приведен в таблице 1.9.

Таблица 1.9 – Обрабатываемые ошибки языка KKV-2020

| Интервал | Описание ошибок |
| --- | --- |
| 0-1 | Системные ошибки |
| 100-104 | Ошибки параметров |
| 110-112 | Ошибки открытия и чтения файлов |
| 113 - 131 | Ошибка лексического анализа |
| 600-606 | Ошибки синтаксического анализа |
| 700-712 | Ошибки семантического анализа |

## **Контрольный пример**

Контрольный пример представлен в Приложении А.

# **Глава 2. Структура транслятора**

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

Основными компонентами транслятора являются лексический анализатор, синтаксический анализатор, семантический анализатор и генератор кода, приведенные на рисунке 2.1.

Транслятор – это программа преобразующая исходный код на одном языке программирования в исходный код на другом языке программирования.

Лексический анализатор – принимает на вход уже первично обработанный и разбитый на отдельные компоненты исходный код на языке KKV-2020. Формирует таблицу идентификаторов и таблицу лексем, а также занимается обнаружением ошибок, связанных с лексикой языка.

Синтаксический анализатор – принимает на вход таблицу лексем, сформированную лексическим анализатором. Перебирая каждое правило языка (допустимую конструкцию) он выявляет синтаксические ошибки, допущенные в исходном коде. Формирует дерево разбора, а также выводит трассировку (разбор) цепочек.[3]

Семантический анализатор – состоит из нескольких функций, отвечающих за выявления тех или иных ошибок, а также некоторых проверок, выполняемых на этапе лексического анализатора. В зависимости от задачи функции на ее вход подается таблица лексем либо таблица лексем и идентификаторов.

Генератор кода – принимает на вход таблицу идентификаторов и таблицу лексем. Задача этого компонента заключается в трансляции, уже пройденного все предыдущие этапы кода на языке KKV-2020, в код на языке Ассемблер.

Структура транслятора представлена на рисунке 2.1.

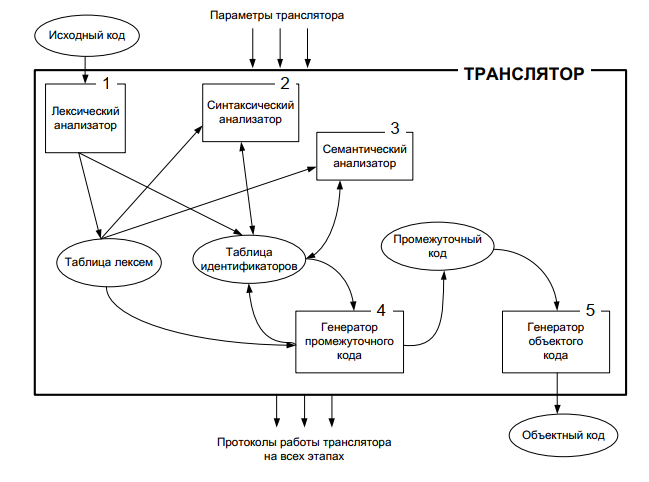


Рисунок 2.1 – Структура транслятора

## **2.2 Перечень входных параметров транслятора**

Входные параметры транслятора представлены в таблице 2.1.

Таблица 2.1 Входные параметры транслятора языка KKV-2020

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<имя\_файла> | Входной файл с расширением .txt, в котором содержится исходный код языка KKV-2020 | Не предусмотрено |
| -log:<имя\_файла> | Файл, содержащий информацию о работе транслятора. | <имя\_файла>.log |
| -out:<имя\_файла> | Файл для записи результата работы лексического и синтаксического анализаторов. | <имя\_файла>.out |
| -lextable | Запись таблицы лексем в файл лога | False |
| -idtable | Запись таблицы идентификаторов в файл лога | False |

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

Протоколы транслятора KKV-2020 представлены в таблице 2.2.

Таблица 2.2 - Протоколы транслятора KKV-2020

|  |  |
| --- | --- |
| Протокол | Содержимое |
| <имя\_файла>.txt | Содержит служебную информацию, получаемую в ходе работы транслятора, а так же ошибки, которые возникают на этапе обработки исходного кода. |

Протокол работы нужен для отображения хода выполнения трансляции языка KKV-2020. Благодаря им пользователь может обнаружить некорректно введенные данные или ошибки в исходном коде программы.

**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 последовательность символов входного языка. Лексический анализатор выделяет в этой последовательности простейшие конструкции языка, которые называют лексическими единицами. 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. Результатом работы лексического анализатора являются таблица лексем и идентификаторов. Структура лексического анализатора представлена на рисунке 3.1.



Рисунок 3.1 ­­­– Структура лексического анализатора KKV-2020

## **3.2 Контроль входных символов**

На рисунке 3.2 представлена таблица входных символов.

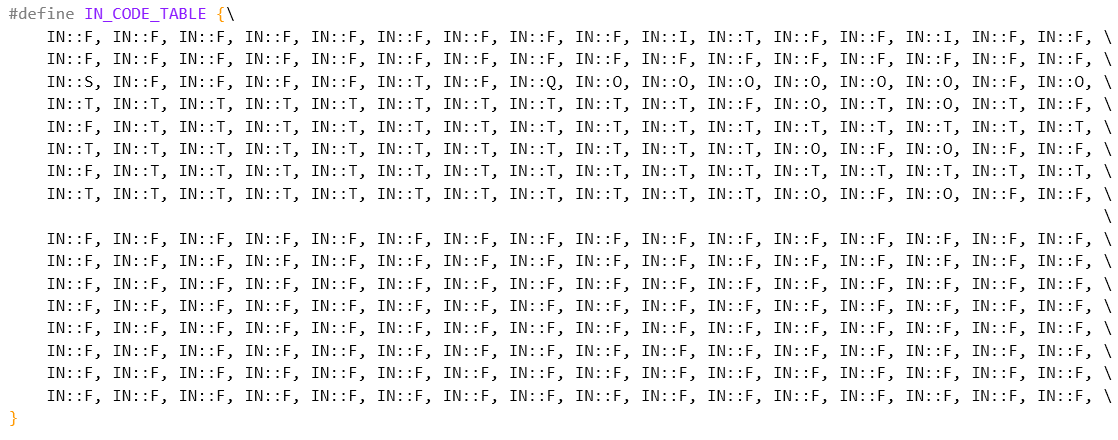


Рисунок 3.2. Таблица контроля входных символов

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, I – игнорируемый символ, O – символ операции, Q – символ кавычки.

## **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, 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. Соответствие цепочек и лексем представлено в таблице 3.1.

Таблице 3.1 – Таблица соответствий цепочек, их типов и лексем

| Цепочка | Тип | Лексема |
| --- | --- | --- |
| Concat | Ключевое слово | s |
| Copy | Ключевое слово | c |
| dec | Ключевое слово | t |
| str | Ключевое слово | t |
| Идентификатор | Переменная, функция или параметр | i |
| Целочисленный литерал | Данные | l |
| Строковый  литерал | Данные | l |
| main | Ключевое слово | m |
| func | Ключевое слово | f |
| new | Ключевое слово | d |
| ret | Ключевое слово | r |
| console | Ключевое слово | p |
| while | Ключевое слово | w |
| +  -  \*  /  %  = | Операторы | +  -  \*  /  %  = |
| ;  ,  []  {}  () | Сепараторы | ;  ,  []  {}  () |

Пример реализации таблицы лексем представлен в приложении А. Также в приложении А находятся конечные автоматы, соответствующие лексемам языка KKV-2020.

## **3.5 Основные структуры данных**

Основные структуры таблиц лексем и идентификаторов данных языка KKV-2020, используемых для хранения, представлены в приложении А. В таблице лексем содержится лексема, её номер, полученный при разборе, номер строки в исходном код. В таблице идентификаторов содержится имя идентификатора, имя литерала, номер в таблице лексем, тип данных, смысловой тип идентификатора и его значение.

## **3.6 Принцип обработки ошибок**

В случае если в результате работы лексического анализатора найдена ошибка, то анализатор заполняет структуру, содержащую ошибки, произошедшие в процессе лексического анализа. После чего вызывается функция генерации ошибки, в которую передается, в зависимости от места возникновения ошибки, следующая информация: код ошибки, номер строки в коде, номер позиции в строке или только код ошибки. Пользователь может ознакомиться с данной ошибкой, открыв протокол.

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

Сообщение лексического анализатора представлено на рисунке 3.3.

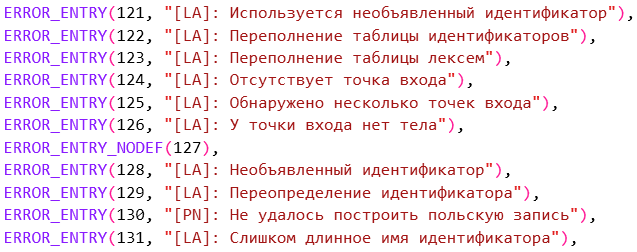


Рисунок 3.3 – Ошибка лексического анализатора

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

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

## **3.9 Алгоритм лексического анализа**

Лексический анализатор проверяет входной поток символов программы на исходном коде на допустимость, удаляет лишние пробелы. Для выделенной части входного потока выполняется функция распознавания лексемы. При успешном распознавании информация о выделенной лексеме заносится в таблицу лексем и таблицу идентификаторов, и алгоритм возвращается к началу. При неуспешном распознавании выдается сообщение об ошибке, а дальнейшие действия зависят от реализации анализатора – либо его выполнение прекращается, либо делается попытка распознать следующую лексему (идет возврат к началу алгоритма).

## **3.10 Контрольный пример**

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

# **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. Входом для синтаксического анализа является таблица лексем и таблица идентификаторов, полученные после фазы лексического анализа. Выход – дерево разбора. Структура синтаксического анализатора представлена на рисунке 4.1.

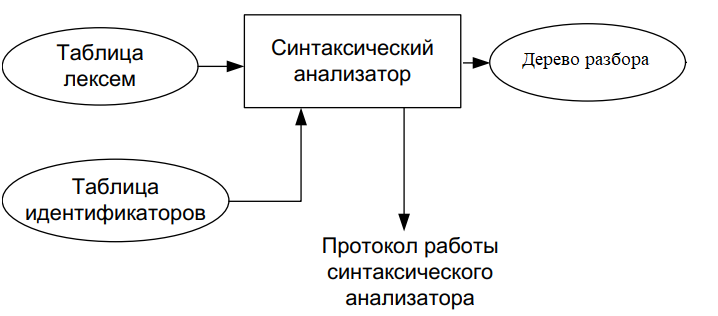


Рисунок 4.1 – Структура синтаксического анализатора

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

В синтаксическом анализаторе транслятора языка KKV-2020 используется контекстно-свободная грамматика , где

T – множество терминальных символов,

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 KKV-2020 представлена в приложении Б.

TS – терминальные символы, NS – нетерминальные символы.

Таблица 4.1 – Перечень правил, составляющих грамматику языка и описание нетерминальных символов KKV-2020

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| S | m{NrE;}; tfi(F){NrE;};S m{NrE;};S | Правила для общих структур |
| N | dti;  rE;  i=E;  dtfi(F);  dti;N  rE;N  i=E;N  dtfi(F);N  c(W);  c(W);N  w(i)[N]; w(i)[N];N | Правила для конструкций и инструкций |
| E | i  l  (E)  i(W)  iM  lM  (E)M  i(W)M | Правила для выражений |
| M | vE  vEM  v(E)  v(E)M | Правила для операторов |
| F | ti  ti,F | Правила для параметров функции |
| W | i  l  i,W  l,W | Правила для параметров вызываемой функции |

## **4.3 Построение конечного магазинного автомата**

В данном курсовом проекте грамматика приведена к нормальной форме Грейбах. Это означает, что каждое правило имеет вид A → aα, где a∈T, α∈N. Конечный автомат с магазинной памятью можно представить в виде семёрки , где М – автомат, Q – множество состояний, V – алфавит входных символов, Z – алфавит магазина,  - функция переходов,  – начальное состояние автомата,  – начальное состояние магазина, F – множество конечных состояний.

## **4.4 Основные структуры данных**

Основные структуры данных синтаксического анализатора включают в себя структуру магазинного автомата и структуру грамматики Грейбах, описывающей правила языка KKV-2020. Данные структуры представлены в приложении Б.

## **4.5 Описание алгоритма синтаксического разбора**

Алгоритм синтаксического разбора:

1. Поиск и выделение синтаксических конструкций в исходном тексте (разбор).
2. Распознавание (проверка правильности) синтаксических конструкций.
3. Выявление ошибок и продолжение процесса распознавания после обработки ошибок.
4. Если нет ошибок, формирование дерева разбора.

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

Перечень сообщений синтаксического анализатора представлен на рисунке 4.3.

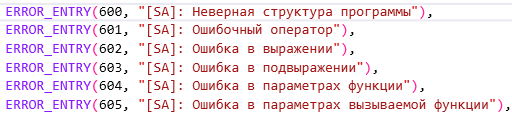


Рисунок 4.3 – Перечень сообщений синтаксического анализатора

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

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

Выходным параметром является дерево разбора, которое записывается в выходной файл.

## **4.8 Принцип обработки ошибок**

Синтаксический анализатор перебирает всевозможные правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем. Если не была найдена ни одна подходящая цепочка, то формируется соответствующая ошибка. Пользователь может ознакомиться с данной ошибкой, открыв протокол.

## **4.9 Контрольный пример**

Пример разбора синтаксическим анализатором исходного кода на языке KKV-2020 представлен в приложении Б. Дерево разбора исходного кода также представлено в приложении Б.

# **Глава 5. Разработка семантического анализатора**

## **5.1 Структура семантического анализатора**

Семантический анализ происходит после выполнения синтаксического анализа и реализуется в виде отдельных проверок таблиц литералов и идентификаторов на соответствие правилам.

## **5.2 Функции семантического анализатора**

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 5.1.

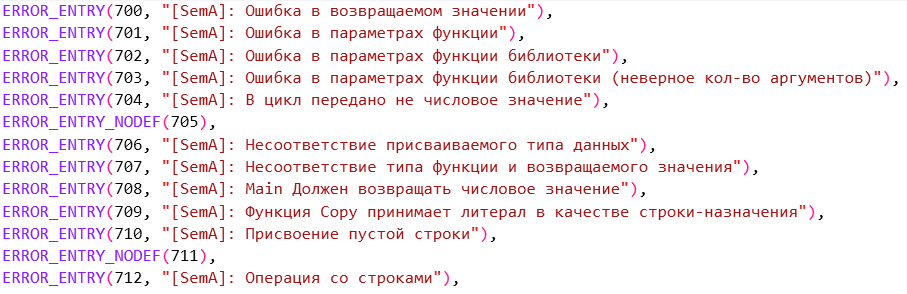


Рисунок 5.1 – Перечень сообщений семантического анализатора

## **5.4 Принцип обработки ошибок**

В качестве входных параметров выступают таблица литералов и идентификаторов. Далее происходит полный анализ данных таблиц. В случае возникновения ошибок, вызываем функцию получения ошибки, которая принимает обязательным параметром код ошибки в таблице сообщений, номер строки и позицию. Пользователь может ознакомиться с данной ошибкой, открыв протокол.

## **5.5 Контрольный пример**

Результат работы контрольного примера расположен в приложении В.

# **Глава 6. Преобразование выражений**

## **6.1 Выражения, допускаемые языком**

Выражения и операции, допускаемые языком, подробно описаны в разделе 1.12 и 1.13.

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

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

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

Алгоритм построения:

– 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 |
| (x+y)\*z |  |  |
| x+y)\*z |  | ( |
| +y)\*z | x |  |
| y)\*z | x | ( + |
| )\*z | x y | ( + |
| \*z | x y + |  |
| z | x y + | \* |
|  | x y + z | \* |
|  | x y + z \* |  |

**6.3 Программная реализация обработки выражений**

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

## **6.4 Контрольный пример**

В приложении Г приведены изменённая таблица лексем, отображающая результаты преобразования выражений в польский формат.

# **Глава 7. Генерация кода**

## **7.1 Структура генератора кода**

Генерация кода – заключительный этап работы транслятора. Результатом данного этапа будет код, сгенерированный для выполнения на Ассемблере на основе таблицы лексем и таблицы идентификаторов, что является требуемым результатом работы программы.Транслятор кода начинает свою работу только в том случае если код на языке KKV-2020 прошёл предыдущие компоненты транслятора без ошибок. Схема данного этапа изображена 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 KKV-2020 размещены в сегменте данных(.data). Литералы – в сегменте констант (.const). Соответствия между типами данных идентификаторов на языке KKV-2020 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка KKV-2020 и языка ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке KKV-2020 | Тип идентификатора на языке ассемблера | Пояснение |
| dec | DWORD | Хранит целочисленный тип данных без знака (4 байта). |
| str | DWORD | Хранит один символ (1 байт). |
| L(0-9) | BYTE  DWORD | Литералы: строковые,  целочисленные |

## **7.3 Алгоритм работы генератора кода**

Генерация кода происходит на основе таблицы лексем и идентификаторов. Каждый сегмент кода Ассемблера описывается отдельно(.const, .data, .code). В сегменте .const описываются литералы, в .data – идентификаторы, а в .code описываются конструкции. Сегмент .code разделен на 2 части: описание главной функции и локальных функций. Каждая строка кода описывается блоками. Результат преобразования записывается в файл с расширением .asm, который расположен в готовом проекте.

## **7.4 Состав статической библиотеки**

Все функции статической библиотеки описаны в разделе 1.18.

## **7.5 Контрольный пример**

Результат работы генерации кода представлен в приложении Д.

# **Глава 8. Тестирование транслятора**

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

В языке KKV-2020 не разрешается использовать запрещённые входным алфавитом символы. Результат использования запрещённого символа показан в таблице 8.1.

Таблица 8.1 – Тестирование фазы проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| new dec П; | Ошибка 111:Недопустимый символ в исходном коде (-in), строка 0, позиция 8. |
| new dec .; | Ошибка 111 Недопустимый символ в исходном файле (-in), позиция 8, строка 0 |

## **8.2 Тестирование лексического анализатора**

На этапе лексического анализа могут возникнуть ошибка, описанная в пункте 3.7. Результаты тестирование лексического анализатора показано в таблице 8.2.

Таблица 8.2 – Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| new dec x;  console y; | y - 1 ---> Ошибка 128 [LA]: Необъявленный идентификатор |
| main{}  main{} | Ошибка 125 [LA]: Обнаружено несколько точек входа |
| main | Ошибка 126 [LA]: У точки входа нет тела |
| new dec x;  new dec x; | x - 1 ---> Ошибка 129 [LA]: Переопределение идентификатора |

## **8.3 Тестирование синтаксического анализатора**

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

Таблица 8.3 – Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| new dec a | Ошибка 601: Ошибочный оператор, строка 0 |
| new dec k;  k = k++; | 602: строка 8, [SA]: Ошибка в выражении |

## **8.4 Тестирование семантического анализатора**

Итоги тестирования семантического анализатора приведены в таблице 8.4.

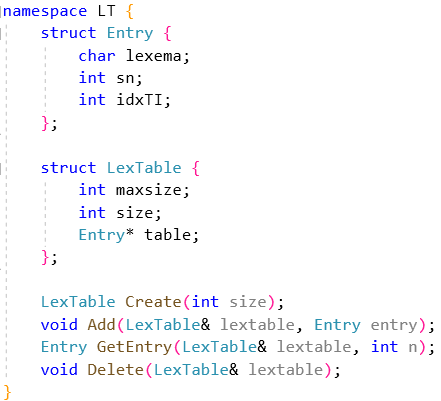
Таблица 8.4 – Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| new dec x;  x = ‘asd’ | Ошибка 706: Несоответствие присваемого типа данных, строка 32 |

# ПРИЛОЖЕНИЕ А

Контрольный пример:

|  |
| --- |
| dec func Sum(dec a, dec b){  new dec sum;  sum = a + b;  ret sum;  }  main{  new dec a;  new dec b;  new dec c;  new dec out;  a = 10 \* 0101;  b = 1010 / 10;  c = 1010 % 11;  out = Sum(b, c);  console a; #1010  console b; #101  console c;  console out;  new str sa;  new str sb;  sa = 'Hello, ';  sb = 'World!';  new func Concat(str str1, str str2);  new str concatenated;  concatenated = Concat(sa, sb);  console concatenated;  new func Copy(str origin, str destination);  Copy('asd', sb);  console sb;  new dec iterator;  iterator = 1010;  while (10)[  console iterator;  ];  ret 0;  } |

Структуры данных лексического анализатора: 

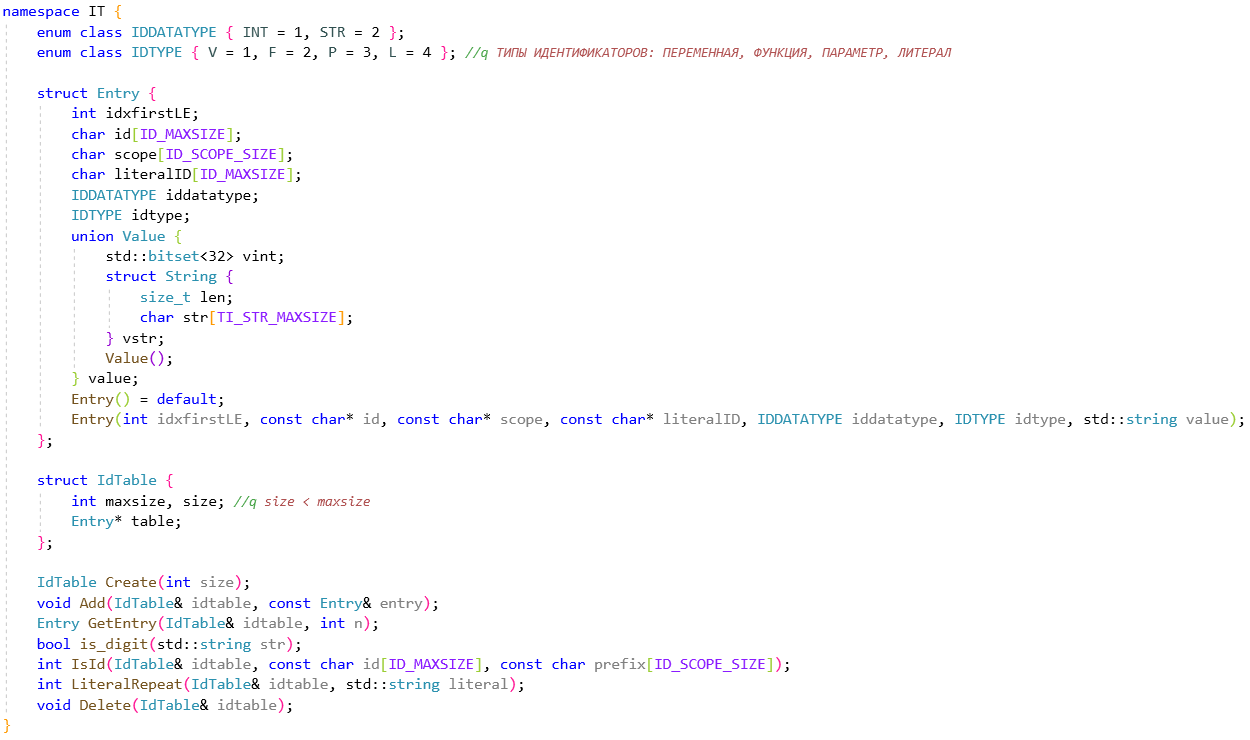


Таблица лексем:

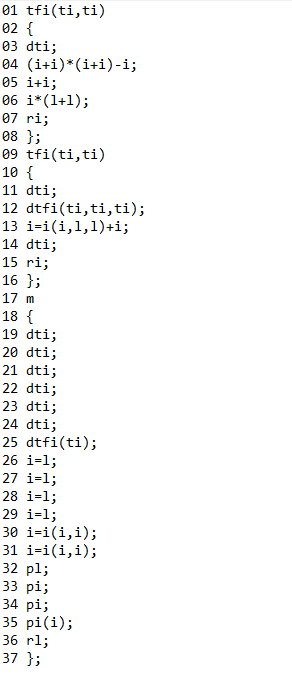


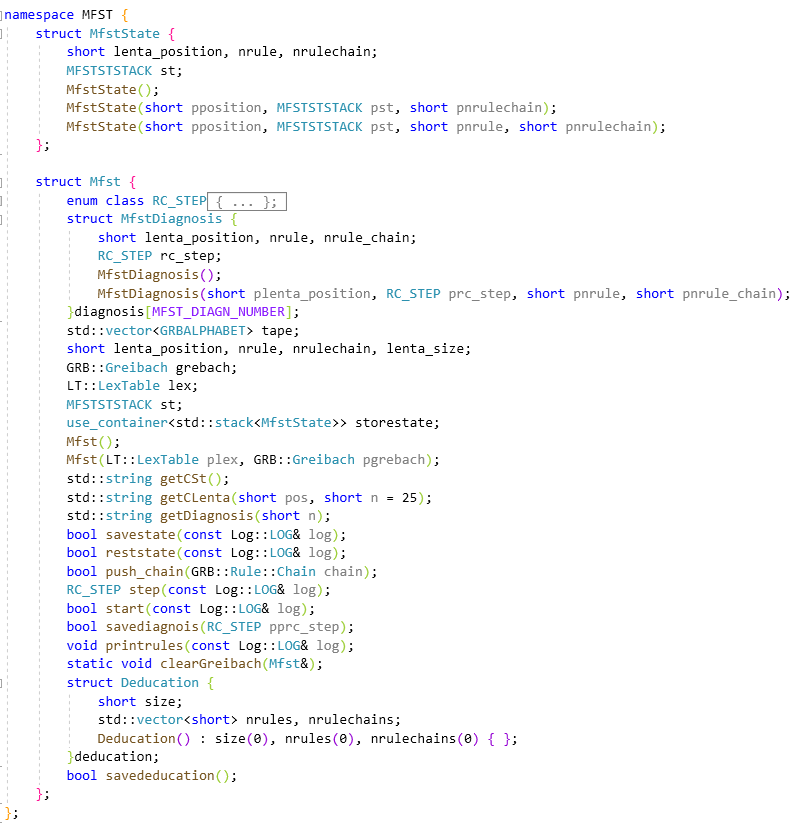
Таблица идентификаторов:

|  |
| --- |
| 0 ----> - Sum - INT - FUNCTION - 2 - 00000000000000000000000000000000  1 ----> Sum - a - INT - PARAMETER - 5 - 00000000000000000000000000000000  2 ----> Sum - b - INT - PARAMETER - 8 - 00000000000000000000000000000000  3 ----> Sum - sum - INT - VARIABLE - 13 - 00000000000000000000000000000000  4 ----> - main - INT - FUNCTION - 25 - 00000000000000000000000000000000  5 ----> main - a - INT - VARIABLE - 29 - 00000000000000000000000000000000  6 ----> main - b - INT - VARIABLE - 33 - 00000000000000000000000000000000  7 ----> main - c - INT - VARIABLE - 37 - 00000000000000000000000000000000  8 ----> main - out - INT - VARIABLE - 41 - 00000000000000000000000000000000  9 ----> l0 - INT - LITERAL - 45 - 00000000000000000000000000000010  10 ----> l1 - INT - LITERAL - 47 - 00000000000000000000000000000101  11 ----> l2 - INT - LITERAL - 51 - 00000000000000000000000000001010  12 ----> l3 - INT - LITERAL - 59 - 00000000000000000000000000000011  13 ----> main - sa - STR - VARIABLE - 84 -  14 ----> main - sb - STR - VARIABLE - 88 -  15 ----> l4 - STR - LITERAL - 92 - 'Hello, '  16 ----> l5 - STR - LITERAL - 96 - 'World!'  17 ----> - Concat - STR - FUNCTION - 100 -  18 ----> Concat - str1 - STR - PARAMETER - 103 -  19 ----> Concat - str2 - STR - PARAMETER - 106 -  20 ----> main - concatenated - STR - VARIABLE - 111 -  21 ----> - Copy - INT - FUNCTION - 127 - 00000000000000000000000000000000  22 ----> Copy - origin - STR - PARAMETER - 130 -  23 ----> Copy - destination - STR - PARAMETER - 133 -  24 ----> l6 - STR - LITERAL - 138 - 'asd'  25 ----> main - iterator - INT - VARIABLE - 148 - 00000000000000000000000000000000  26 ----> l7 - INT - LITERAL - 165 - 00000000000000000000000000000000 |

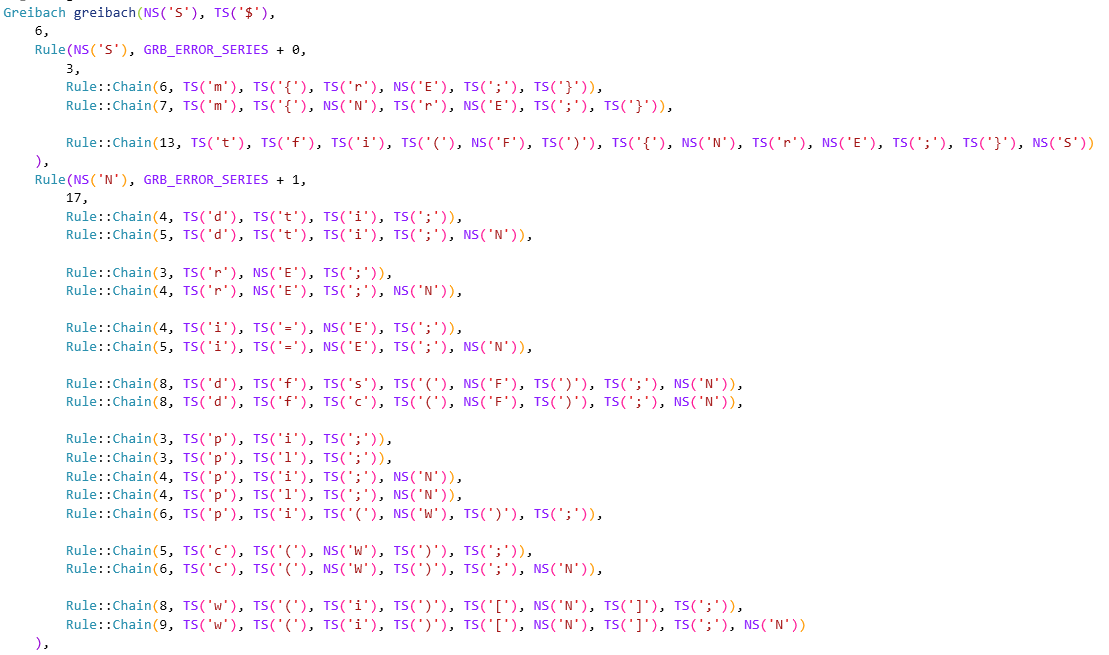
# **Приложение Б**

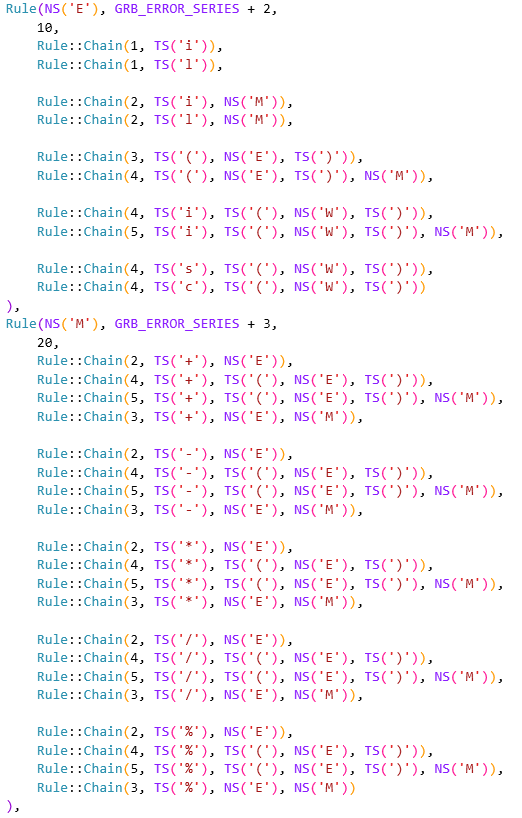
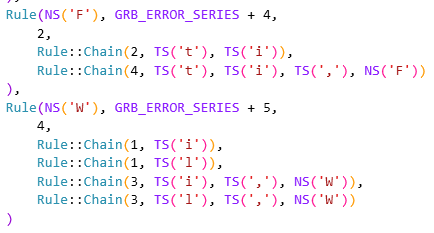
Структуры данных синтаксического анализатора:





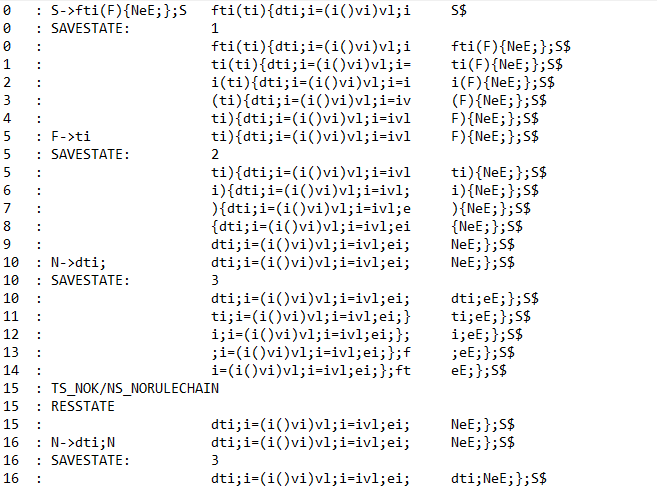
Грамматика Грейбах:



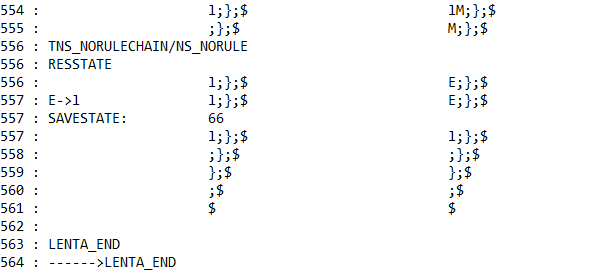


Дерево разбора:

* Начало:



* Конец:



Правила разбора:

|  |
| --- |
| 0 : S->tfi(F){NrE;}S  4 : F->ti,F  7 : F->ti  11 : N->dti;N  15 : N->i=E;  17 : E->iM  18 : M->+E  19 : E->i  22 : E->i  25 : S->m{NrE;}  27 : N->dti;N  31 : N->dti;N  35 : N->dti;N  39 : N->dti;N  43 : N->i=E;N  45 : E->lM  46 : M->\*E  47 : E->l  49 : N->i=E;N  51 : E->lM  52 : M->/E  53 : E->l  55 : N->i=E;N  57 : E->lM  58 : M->%E  59 : E->l  61 : N->i=E;N  63 : E->i(W)  65 : W->i,W  67 : W->i  70 : N->pi;N  73 : N->pi;N  76 : N->pi;N  79 : N->pi;N  82 : N->dti;N  86 : N->dti;N  90 : N->i=E;N  92 : E->l  94 : N->i=E;N  96 : E->l  98 : N->dfs(F);N  102 : F->ti,F  105 : F->ti  109 : N->dti;N  113 : N->i=E;N  115 : E->s(W)  117 : W->i,W  119 : W->i  122 : N->pi;N  125 : N->dfc(F);N  129 : F->ti,F  132 : F->ti  136 : N->c(W);N  138 : W->l,W  140 : W->i  143 : N->pi;N  146 : N->dti;N  150 : N->i=E;N  152 : E->l  154 : N->w(i)[N];  159 : N->pi;  165 : E->l |

# **Приложение В**

Ошибки семантического анализатора:

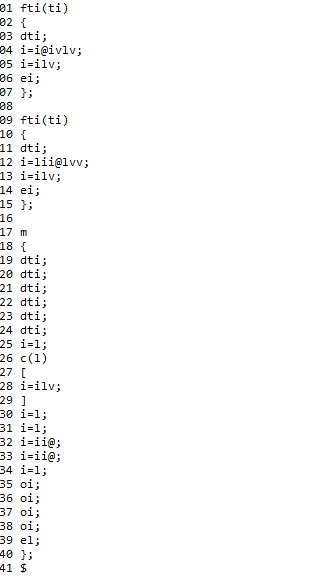
|  |
| --- |
| Ошибка 700 [SemA]: Ошибка в возвращаемом значении , строка 3 |
| Ошибка 701 [SemA]: Ошибка в параметрах функции , строка 15 |
| Ошибка 702 [SemA]: Ошибка в параметрах функции библиотеки , строка 27 |
| Ошибка 703 [SemA]: Ошибка в параметрах функции библиотеки (неверное кол-во аргументов) , строка 27 |
| Ошибка 704 [SemA]: В цикл передано не числовое значение , строка 37 |
| Ошибка 706 [SemA]: Несоответствие присваиваемого типа данных , строка 22 |
| Ошибка 707 [SemA]: Несоответствие типа функции и возвращаемого значения , строка 3 |
| Ошибка 708 [SemA]: Main Должен возвращать числовое значение , строка 41 |
| Ошибка 709 [SemA]: Функция Copy принимает литерал в качестве строки-назначения , строка 31 |
| Ошибка 710 [SemA]: Присвоение пустой строки , строка 22 |
| Ошибка 712 [SemA]: Операция со строками , строка 25 |

# **Приложение Г**

Программная реализация обработки выражений:

|  |
| --- |
| bool PN::Convertation(int lextable\_pos, LT::LexTable& lextable, IT::IdTable& idtable)  {  container<std::stack<char>> stack;  std::string PolishString;  std::vector<char> operators = { LEX\_MINUS, LEX\_PLUS, LEX\_DIRSLASH, LEX\_STAR, LEX\_PERCENT };  std::vector<int> ids;  int operators\_count = 0, operands\_count = 0, iterator = 0, right\_counter = 0, left\_counter = 0, params\_counter = 0;  for (int i = lextable\_pos; i < lextable.size; i++, iterator++) {  char lexem = lextable.table[i].lexema;  size\_t stack\_size = stack.size();  if (idtable.table[lextable.table[i].idxTI].idtype == IT::IDTYPE::F) {  stack.push(LEX\_HEADOFFUNC);  operands\_count--;  }  if (std::find(operators.begin(), operators.end(), lexem) != operators.end()) {  if (!stack.empty() && stack.top() != LEX\_LEFTHESIS) {  while (!stack.empty() && PN::get\_priority(lexem) <= PN::get\_priority(stack.top())) {  PolishString += stack.top();  stack.pop();  }  }  stack.push(lexem);  operators\_count++;  }  else if (lexem == LEX\_COMMA) {  while (!stack.empty()) {  if (stack.top() == LEX\_LEFTHESIS)  break;  PolishString += stack.top();  stack.pop();  }  operands\_count--;  }  else if (lexem == LEX\_LEFTHESIS) {  left\_counter++;  stack.push(lexem);  }  else if (lexem == LEX\_RIGHTHESIS) {  right\_counter++;  if (!PN::find\_elem(stack, stack\_size, LEX\_LEFTHESIS))  return false;  while (stack.top() != LEX\_LEFTHESIS) {  PolishString += stack.top();  stack.pop();  }  stack.pop();  if (!stack.empty() && stack.top() == LEX\_HEADOFFUNC) {  PolishString += stack.top() + To\_string(params\_counter - 1);  params\_counter = 0;  stack.pop();  }  }  else if (lexem == LEX\_SEMICOLON) {  if (operators\_count != 0 && operands\_count != 0)  if ((!stack.empty() && (stack.top() == LEX\_RIGHTHESIS || stack.top() == LEX\_LEFTHESIS))  || right\_counter != left\_counter || operands\_count - operators\_count != 1)  return false;  while (!stack.empty()) {  PolishString += stack.top();  stack.pop();  }  PN::fix\_lextable(lextable, PolishString, iterator, lextable\_pos, ids);  break;  }  else if (lexem == LEX\_ID || lexem == LEX\_LITERAL || lexem == LEX\_CONCAT || lexem == LEX\_COPY) {  if (std::find(stack.c.begin(), stack.c.begin(), LEX\_HEADOFFUNC) != stack.c.end())  params\_counter++;  PolishString += lexem;  if (lextable.table[i].idxTI != LT\_TI\_NULLIDX)  ids.push\_back(lextable.table[i].idxTI);  operands\_count++;  }  }  return true;  } |

Преобразованные выражения:



**Приложение Д**

Результат генерации кода:

|  |
| --- |
| .586  .model flat, stdcall  includelib libucrt.lib  includelib kernel32.lib  includelib ../LP\_LIB/Debug/LP\_Lib.lib  ExitProcess PROTO : DWORD  \_Concat PROTO : DWORD, :DWORD  \_Copy PROTO : DWORD, : DWORD  \_ConsoleWrite PROTO : DWORD  \_ConsoleWriteInt PROTO : DWORD  .stack 4096  .const  \_DIVISION\_BY\_ZERO\_ERROR BYTE 'Ошибка выполнения: деление на ноль', 0  l0 DWORD 00000000000000000000000000000010y  l1 DWORD 00000000000000000000000000000101y  l2 DWORD 00000000000000000000000000001010y  l3 DWORD 00000000000000000000000000000011y  l4 BYTE 'Hello, ', 0  l5 BYTE 'World!', 0  l6 BYTE 'asd', 0  l7 DWORD 00000000000000000000000000000000y  .data  \_Sumsum DWORD 0 ;INT  \_maina DWORD 0 ;INT  \_mainb DWORD 0 ;INT  \_mainc DWORD 0 ;INT  \_mainout DWORD 0 ;INT  \_mainsa DWORD 0 ;STR  \_mainsb DWORD 0 ;STR  \_mainconcatenated DWORD 0 ;STR  \_mainiterator DWORD 0 ;INT  .code  \_Sum PROC \_b: DWORD, \_a: DWORD  push \_a  push \_b  ;\/Сложение\/  pop eax  pop ebx  add eax, ebx  push eax  ;/\Сложение/\  pop \_Sumsum  mov eax, \_Sumsum  ret 8  \_Sum ENDP  main PROC  push l0  push l1  ;\/Умножение\/  pop eax  pop ebx  mul ebx  push eax  ;/\Умножение/\  pop \_maina  push l2  push l0  ;\/Деление\/  pop ebx  mov edx, 0  pop eax  .if ebx == 0  push offset \_DIVISION\_BY\_ZERO\_ERROR  call \_ConsoleWrite  invoke ExitProcess, -1  .endif  div ebx  push eax  ;/\Деление/\  pop \_mainb  push l2  push l3  ;\/Остаток от деления\/  pop ebx  mov edx, 0  pop eax  .if ebx == 0  push offset \_DIVISION\_BY\_ZERO\_ERROR  call \_ConsoleWrite  invoke ExitProcess, -1  .endif  div ebx  push edx  ;/\Остаток от деления/\  pop \_mainc  push \_mainb  push \_mainc  call \_Sum  push eax  pop \_mainout  push \_maina  call \_ConsoleWriteInt  push \_mainb  call \_ConsoleWriteInt  push \_mainc  call \_ConsoleWriteInt  push \_mainout  call \_ConsoleWriteInt  push offset l4  pop \_mainsa  push offset l5  pop \_mainsb  push \_mainsa  push \_mainsb  call \_Concat  push eax  pop \_mainconcatenated  push \_mainconcatenated  call \_ConsoleWrite  push offset \_mainsb  push offset l6  call \_Copy  push \_mainsb  call \_ConsoleWrite  push l2  pop \_mainiterator  .while \_mainiterator  ;\/Тело цикла\/  push \_mainiterator  call \_ConsoleWriteInt  dec \_mainiterator  ;/\Тело цикла/\  .endw  push l7  call ExitProcess  main ENDP  end main |

# **Заключение**

В данном курсовом проекте были выполнены поставленные минимальные требования. В ходе работы было изучено много нового, а также закреплены знания, которые были получены ранее. Также стоит отметить, что данный курсовой проект позволил совместить закрепление знаний сразу по двум языкам программирования, таких как C++ и Ассемблер. При написании приложения были усвоены такие понятия как синтаксический, лексический и семантический анализатор и т.п. В итоге был получен примитивный язык программирования KKV-2020, который не имеет сложных конструкций, которые реализованы на сегодняшний день во многих других языках программирования.

Окончательная версия языка KKV-2020 включает:

1. 2 типа данных;
2. Поддержка оператора вывода;
3. Возможность подключения и вызова функций стандартной библиотеки;
4. Наличие 5 арифметических операторов для вычисления выражений;
5. Возможность вызова функции в выражении;

# **Литература**

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

2. Молчанов, А. Ю. Системное программное обеспечение / А. Ю. Молчанов. – СПб.: Питер, 2010. – 400 с.

3. Ахо, А. Теория синтаксического анализа, перевода и компиляции /А. Ахо, Дж. Ульман. – Москва : Мир, 1998. – Т. 2 : Компиляция. - 487 с.

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

5. Орлов, С.А. Теория и практика языков программирования / С.А. Орлов – 2014. – 689 с.

6. Прата, С. Язык программирования С++. Лекции и упражнения / С. Прата. – М., 2006 — 1104 c.

7. Смелов, В. В. Введение в объектно-ориентированное программирование на С++ / В. В. Смелов.– Минск : БГТУ, 2009. - 94с.

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