МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 Программное обеспечение информацион-

ных технологий

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора WSA-2020»

Выполнил студент Валько Сергей Александрович

(Ф.И.О. студента)

Руководитель проекта ст.пр. Наркевич Аделина Сергеевна

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

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

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

Консультанты ст.пр. Наркевич Аделина Сергеевна

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

Курсовой проект защищен с оценкой

Минск 2020

Оглавление

[Введение 5](#_Toc59015110)

[1.Спецификация языка программирования 6](#_Toc59015111)

[1.1 Характеристика языка программирования 6](#_Toc59015112)

[1.2 Определение алфавита языка программирования 6](#_Toc59015113)

[1.3 Символы сепараторы 6](#_Toc59015114)

[1.4Применяемые кодировки 7](#_Toc59015115)

[1.5 Типы данных 7](#_Toc59015116)

[1.6Преобразование типов 8](#_Toc59015117)

[1.7 Идентификаторы 8](#_Toc59015118)

[1.8 Литералы 8](#_Toc59015119)

[1.9 Объявление данных 9](#_Toc59015120)

[1.10 Инициализация данных 9](#_Toc59015121)

[1.11 Инструкции языка 9](#_Toc59015122)

[1.12 Операции языка 10](#_Toc59015123)

[1.13 Выражения и их вычисление 11](#_Toc59015124)

[1.14 Конструкции языка 11](#_Toc59015125)

[1.15 Область видимости идентификаторов 12](#_Toc59015126)

[1.16 Семантические проверки 12](#_Toc59015127)

[1.17Распределение оперативной памяти на этапе выполнения 13](#_Toc59015128)

[1.18 Стандартная библиотека и её состав 13](#_Toc59015129)

[1.19 Ввод и вывод данных 13](#_Toc59015130)

[1.20 Точка входа 13](#_Toc59015131)

[1.21 Препроцессор 13](#_Toc59015132)

[1.22 Соглашения о вызовах 14](#_Toc59015133)

[1.23 Объектный код 14](#_Toc59015134)

[1.24 Классификация сообщений транслятора 14](#_Toc59015135)

[1.25 Контрольный пример 14](#_Toc59015136)

[2. Структура транслятора 15](#_Toc59015137)

[2.1. Компоненты транслятора их назначение и принципы взаимодействия 15](#_Toc59015138)

[2.2 Перечень входных параметров транслятора 16](#_Toc59015139)

[2.3 Протоколы, формируемые транслятором 16](#_Toc59015140)

[3.Разработка лексического анализатора 17](#_Toc59015141)

[3.1 Структура лексического анализатора 17](#_Toc59015142)

[3.2 Контроль входных символов 17](#_Toc59015143)

[3.3 Удаление избыточных символов 18](#_Toc59015144)

[3.4 Перечень ключевых слов 19](#_Toc59015145)

[3.5 Основные структуры данных 21](#_Toc59015146)

[3.6 Структура и перечень сообщений лексического анализатора 21](#_Toc59015147)

[3.7 Принцип обработки ошибок 22](#_Toc59015148)

[3.8 Параметры лексического анализатора 22](#_Toc59015149)

[3.9 Алгоритм лексического анализа 22](#_Toc59015150)

[3.10 Контрольный пример 23](#_Toc59015151)

[4.Разработка синтаксического анализатора 24](#_Toc59015152)

[4.1 Структура синтаксического анализатора 24](#_Toc59015153)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 24](#_Toc59015154)

[4.3 Построение конечного магазинного автомата 26](#_Toc59015155)

[4.4 Основные структуры данных 27](#_Toc59015156)

[4.5 Описание алгоритма синтаксического разбора 27](#_Toc59015157)

[Принцип работы автомата следующий: 27](#_Toc59015158)

[4.6 Структура и перечень сообщений синтаксического анализатора 27](#_Toc59015159)

[4.7. Параметры синтаксического анализатора и режимы его работы 28](#_Toc59015160)

[4.8. Принцип обработки ошибок 28](#_Toc59015161)

[4.9. Контрольный пример 28](#_Toc59015162)

[5. Разработка семантического анализатора 29](#_Toc59015163)

[5.1 Структура семантического анализатора 29](#_Toc59015164)

[5.2 Функции семантического анализатора 29](#_Toc59015165)

[5.3 Структура и перечень сообщений семантического анализатора 29](#_Toc59015166)

[5.4 Принцип обработки ошибок 29](#_Toc59015167)

[5.5 Контрольный пример 30](#_Toc59015168)

[6. Вычисление выражений 31](#_Toc59015169)

[6.1 Выражения, допускаемые языком 31](#_Toc59015170)

[6.2 Польская запись и принцип её построения 31](#_Toc59015171)

[6.3 Программная реализация обработки выражений 32](#_Toc59015172)

[6.4 Контрольный пример 32](#_Toc59015173)

[7. Генерация кода 33](#_Toc59015174)

[7.1 Структура генератора кода 33](#_Toc59015175)

[7.2 Представление типов данных в оперативной памяти 33](#_Toc59015176)

[7.3 Статическая библиотека 34](#_Toc59015177)

[7.4 Особенности алгоритма генерации кода 35](#_Toc59015178)

[7.5 Входные параметры генератора кода 35](#_Toc59015179)

[7.6 Контрольный пример 35](#_Toc59015180)

[8. Тестирование транслятора 36](#_Toc59015181)

[8.1 Общие положения 36](#_Toc59015182)

[8.2 Результаты тестирования 36](#_Toc59015183)

[Заключение 38](#_Toc59015184)

[Список использованных источников 39](#_Toc59015185)

[Приложение А 40](#_Toc59015186)

[Приложение Б 43](#_Toc59015187)

[Приложение В 45](#_Toc59015188)

[Приложение Г 50](#_Toc59015189)

[Приложение Д 54](#_Toc59015190)

# Введение

Язык программирования WSA-2020 является процедурным, строго типизированным, компилируемый.

Состав транслятора:

* лексический анализатор;
* синтаксический анализатор;
* семантический анализатор;
* генератор кода, или интерпретатор.

Все части транслятора, взаимодействуя между собой, обрабатывают входной текст и строят для него эквивалентный текст на понятном компьютеру языке программирования (ассемблере).

Задача курсового проекта: разработать собственный язык программирования, а также транслятор для него. Язык получил название WSA-2020. Задача разбивается на этапы:

* разработка спецификации языка WSA-2020;
* разработка структуры транслятора;
* разработка лексического анализатора;
* разработка синтаксического анализатора;
* разработка семантического анализатора;
* преобразование выражений;
* генерация кода:
* тестирование транслятора.

## 1.Спецификация языка программирования

* 1. Характеристика языка программирования

Язык WSA-2020 – это процедурный, строго типизированный, не объектно-ориентированный, компилируемый язык высокого уровня.

* 1. Определение алфавита языка программирования

Алфавит языка WSA-2020 содержит следующие множества:

* прописные и строчные латинские буквы: [a-z, A-Z];
* **прописные и строчные буквы кириллицы: [а-я, А-Я];**
* цифры: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
* специальные символы: {\* + - ? ‘ ,: ; $ # = % ( ) { } < > !};
* пробельный символ;
* символ перехода на новую строку.

## 1.3 Символы сепараторы

Символы–сепараторы представлены в таблице 1.1.

Таблица 1.1 Символы-сепараторы

|  |  |
| --- | --- |
| Символ-сепаратор | Характеристика |
| ; | Разделяет конструкции языка. |
| Пробел | Допускается везде кроме идентификаторов и ключевых слов. |
| () | Определяют приоритет операций. Содержат параметры функции или процедуры. |
| {} | Программный блок. |

* 1. Применяемые кодировки

В языке WSA-2020 используется кодировка Windows-1251 (CP1251). Таблица кодировки изображена на рисунке 1.1.



Рисунок 1.1 Таблица кодировки Windows-1251(CP1251).

## 1.5 Типы данных

Язык WSA-1251 поддерживает целочисленный, строковый, символьный и логический типы данных. Описание типов данных, предусмотренных в данном языке, представлено в таблице 1.2.

Таблица 1.2 Типы данных языка программирования WSA-2020

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Тип данных | Размер | Диапазон значений | Значение по умолчанию | Описание |
| short | 2 байта | {-32 768…32 767} | 0 | Фундаментальный тип данных, предназначенный для хранения целых чисел. |
| string | 1 символ -  1 байт | - | "" | Фундаментальный тип данных, предназначенный для представления символьных строк. Максимальная длина строки: 255 символов. |

Продолжение таблицы 1.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Тип данных | Размер | Диапазон значений | Значение по умолчанию | Описание |
| bool | 1 байт | true и false | false | Фундаментальный тип данных, содержащий значения: true(истина) и false(ложь). |
| char | 1байт | {-128 … 127} | '' | Фундаментальный тип данных, предназначенный для хранения символьной информации |

* 1. Преобразование типов

В языке WSA-2020 отсутствует преобразование типов.

## 1.7 Идентификаторы

В имени идентификатора используются символы латинского алфавита нижнего регистра и цифры (цифра не должна находиться на первой позиции). Максимальная длина идентификатора 8 символов. В случае, если эта длина превышена, имя идентификатора урезается. Пример, верно, записанного идентификатора: value123. Пример, неверно записанного идентификатора: 12val.

## 1.8 Литералы

В языке WSA-2020 присутствует 4 типа литералов: логические, целочисленные (двоичное и восьмеричное представление), строковые и символьные. Краткое описание литералов представлено в таблице 1.3.

Таблица 1.3 Описание литералов

|  |  |
| --- | --- |
| Литерал | Описание |
| Целочисленный | Восьмеричное представление: 0oX или 1oX, X – число в восьмеричной системе счисления .  Двоичное представление: 0bX или 1bX, X - число в двоичной системе счисления.  (0 – положительное число, 1 – отрицательное число)  Диапазон значений: от -32 768 до 32 767. |

Продолжение таблицы 1.3

|  |  |
| --- | --- |
| Литерал | Описание |
| Строковый | Символы, помещённые в "" (код в Windows-1251: 34).  Максимальное количество символов: 255. |
| Логические | true (значение - 1)  false (значение - 0) |
| Символьные | Символ, помещённый в ' ' (код в Windows-1251: 39). |

Пример целочисленного литерала: 0b111001, 0o1234.Пример строкового литерала: ' dggfgfdhd '.

## 1.9 Объявление данных

Конструкция объявления переменной: var <тип данных> <идентификатор>;

## 1.10 Инициализация данных

Язык WSA-2020 не поддерживает инициализацию данных. Но язык предусматривает инициализацию по умолчанию (сведения об инициализации по умолчанию для конкретных типов данных предоставлены в таблице 1.2).

## 1.11 Инструкции языка

Инструкции языка программирования WSA-2020 представлены в таблице 1.4.

Таблица 1.4 Инструкции языка программирования WSA-2020

|  |  |
| --- | --- |
| Инструкция | Синтаксис |
| Главная функция | main  {  <инструкции языка>; } |
| Объявление переменной | var <тип данных> <идентификатор>; |
| Объявление функции | func <тип данных> <идентификатор> (<параметр>, …)  {  <инструкции языка>; } |
| Параметр функции(процедуры) | <тип данных> <идентификатор> |

Продолжение таблицы 1.4

|  |  |
| --- | --- |
| Инструкция | Синтаксис |
| Объявление процедуры | proc<идентификатор> (<параметр>, …)  {  <инструкции языка>; } |
| Операция присваивания | <идентификатор>=<идентификатор>;  <идентификатор>=<литерал>;  <идентификатор>=<выражение>; |
| Вызов функции (процедуры) | $<идентификатор>(<идентификатор >|<литерал>, …); |
| Возврат значения | ret <идентификатор>|<литерал>|<выражение>; |
| Оператор цикла | while(условие)  {  <инструкции языка>; } |
| Вывод данных | write <идентификатор>|<литерал>|<выражение>; |
| Подключение внешней функции | extr <тип данных>  <идентификатор>(<параметр>,…); |

## 1.12 Операции языка

Операции языка программирования WSA-2020 представлены в таблице 1.5.

Таблица 1.5 Операции языка программирования WSA-2020

|  |  |  |  |
| --- | --- | --- | --- |
| Оператор | Тип оператора | Тип данных | Описание |
| + | бинарный | short | Сложение двух целых чисел |
| + | бинарный | string | Конкатенация строк |
| - | унарный | short | Установка знака числа в противоположный |
| - | бинарный | short | Вычитание целых чисел |
| \* | бинарный | short | Умножение целых чисел |
| / | бинарный | short | Деление нацело |
| % | бинарный | short | Деление по модулю |
| < | бинарный | short | Проверяет меньше ли правое число левого |

Продолжение таблицы 1.5

|  |  |  |  |
| --- | --- | --- | --- |
| Оператор | Тип оператора | Тип данных | Описание |
| > | бинарный | short | Проверяет больше ли правое число левого |
| < | бинарный | string | Проверяет меньше ли правая строка левой по длине |
| > | бинарный | string | Проверяет больше ли правая строка левой по длине |
| == | бинарный | short | Проверяет числа на равенство |
| == | бинарный | string | Проверяет строки на равенство по длине |
| == | бинарный | bool | Проверяет логические значения на равенство |
| != | бинарный | bool, string, short | Производит действия обратные оператору ‘==’ |

Приоритет операций (сверху в низ):

1.()

2.== \* / != < > %

3. + -

## 1.13 Выражения и их вычисление

Выражение – выполнение операций над операндами. Операнды могут быть представлены переменными, литералами или вызовом функций. Признаком начала выражения является оператор присваивания, а конца – точка с запятой. Результатом выражения является значение определённого типа. Операнды, над которыми выполняется операция, должны иметь одинаковые типы.

## 1.14 Конструкции языка

Программные конструкции языка WSA-2020 представлены в таблице 1.6.

Таблица 1.6 Конструкции языка программирования WSA-2020

|  |  |
| --- | --- |
| Конструкция | Синтаксис |
| Главная процедура (точка входа) | main  {  <инструкции языка>; } |
| Функция | <тип данных>  func<идентификатор>(<параметры>)  { <инструкции языка>;  ret <идентификатор>|<литерал>;  } |
| Процедура | proc<идентификатор>(<параметры>)  {  <инструкции языка>; } |

## 1.15 Область видимости идентификаторов

Переменные могут определяться внутри программного блока. Переменные являются локальными. Функции определяются в глобальной области видимости. Объявление пользовательских областей видимости не предусмотрено.

## 1.16 Семантические проверки

В языке WSA-2020 предусмотрены следующие семантические правила:

* переменная или константа должна быть объявлена перед использованием;
* объявленная переменная должно хотя бы раз использоваться в выполняемом коде;
* исходный текст должен обязательно содержать точку входа main;
* результат функции должен быть определён при любом ходе её выполнения;
* оператор цикла должен предусматривать возможность обязательного окончания работы;
* не допускаются идентификаторы, совпадающие с ключевыми словами;
* функция или процедура перед выполнением требует обязательного объявления;
* не разрешается повторное объявление функции или процедуры.

## 1.17Распределение оперативной памяти на этапе выполнения

Транслированный код использует следующие области памяти: сегмент констант и сегмент данных. Сегмент констант содержит литералы, а сегмент данных переменные, параметры функций и возвращаемые значения функции. Локальная область видимости в исходном коде определяется за счет использования правил именования идентификаторов и регулируется их префиксами.

## 1.18 Стандартная библиотека и её состав

Стандартная библиотека языка WSA-2020 представлена в таблице 1.7.

Таблица 1.7 Стандартная библиотека языка программирования WSA-2020

|  |  |  |  |
| --- | --- | --- | --- |
| Идентификатор | Тип возвращаемого значения | Параметры | Описание |
| len | short | string | Возвращает длину полученной строки |
| random | short | short, short | Возвращает случайное число в диапазоне, заданном параметрами функции. |
| writeOct | bool | short | Выводит число в восьмеричном формате. |

Синтаксис подключения функции стандартной библиотеки: extr <тип данных><идентификатор>(<параметры>);

## 1.19 Ввод и вывод данных

В языке WSA-2020 предусмотрен вывод данных в консоль. Вывод осуществляется с помощью ключевого слова ‘write’, его синтаксис: write <идентификатор>|<вызов функции>|<литерал>;

Оператор ввода в языке не предусмотрен.

## 1.20 Точка входа

Точка входа в языке WSA-2020 представлена главной процедурой, которая обозначается ключевым словом ‘main’.

## 1.21 Препроцессор

В языке WSA-2020 препроцессор не предусмотрен.

## 1.22 Соглашения о вызовах

Используется соглашение о вызовах stdcall:

* параметры передаются через стек;
* параметры следуют справа налево;
* стек очищается вызываемым кодом.

## 1.23 Объектный код

Целевой язык трансляции - язык ассемблера для процессоров Intel семейства IA-32.

## 1.24 Классификация сообщений транслятора

Классификации сообщений транслятора содержится в таблице 1.8

Таблица 1.8 Классификация сообщений транслятора

|  |  |
| --- | --- |
| Кодовое значение ошибки | Описание |
| 0-116 | Системные ошибки |
| 117 – 200 | Ошибки на этапе лексического анализа |
| 300 - 400 | Ошибки на этапе синтаксического анализа |
| 600 – 700 | Ошибки на этапе семантического анализа |
| 200-300,400-500, 700 - 1000 | Зарезервированные коды ошибок |

## 1.25 Контрольный пример

Контрольный пример кода на языке WSA-2020 находится в приложении А.

# 2. Структура транслятора

Транслятор преобразует исходный текст программы в текст целевого языка. Получив на вход исходный текст, транслятор проверяет его принадлежность заданному языку и определяет набор грамматических правил языка. Процесс трансляции состоит из фаз: лексический анализ, синтаксический анализ, семантический анализ и генерация кода. На всех фазах трансляции применяется таблица идентификаторов, которая пополняется дополнительной информацией в ходе трансляции.

## 2.1. Компоненты транслятора их назначение и принципы взаимодействия

Транслятор преобразует программу, написанную на языке WSA-2020, в язык ассемблера. Его составные части: лексический анализатор, синтаксический анализатор, семантический анализатор и генератор кода на язык ассемблера. Принцип их взаимодействия представлен на рисунке 2.1.

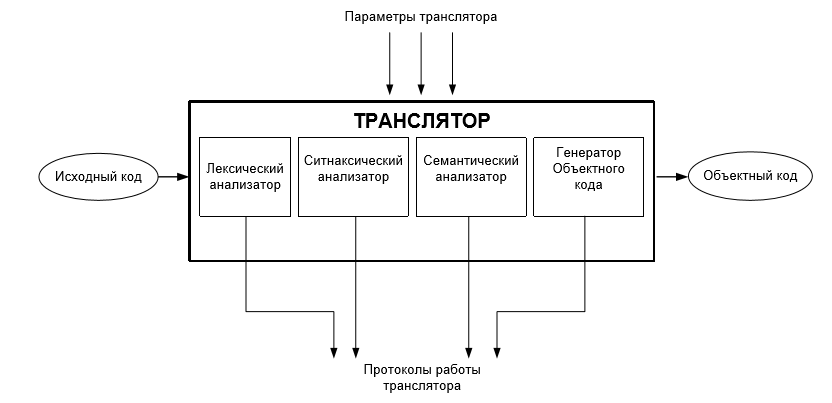
****

Рисунок 2.1 Принцип взаимодействия составных частей транслятора

Лексический анализатор принимает на вход исходный код программы, а по окончании его работы на выходе получаем: таблицу лексем, таблицу идентификаторов и протокол работы. Задача лексического анализатора проверить на правильность входные символы и лексические единицы (лексемы).

Синтаксический анализатор принимает на вход таблицу лексем и идентификаторов, а на выходе выдаёт дерево разбора. Задача синтаксического анализатора проверить правильность цепочек языка.

Семантический анализатор получает таблицу лексем и идентификаторов и проверяет входные данные на соблюдение смысловых нагрузок.

Генератор кода принимает результаты работы предыдущих этапов работы транслятора и на их основе генерирует код на языке ассемблера.

## 2.2 Перечень входных параметров транслятора

Входные параметры транслятора описаны в таблице 2.1.

Таблица 2.1. Перечень входных параметров транслятора

|  |  |  |
| --- | --- | --- |
| Параметр | Характеристика | Значение по умолчанию |
| -in:<путь к файлу> | Описывает путь к исходному файлу на языке WSA-2020. | Не предусмотрено. |
| -log:<путь к файлу> | Описывает путь к log файлу для вывода протоколов работы транслятора. | <имя in-файла>.log |
| -out:<путь к файлу> | Описывает путь к out файлу с результирующим кодом на языке ассемблера. | <имя in-файла>.asm |

## 2.3 Протоколы, формируемые транслятором

Протоколы, формируемые транслятором, описаны в таблице 2.2.

Таблица 2.2 Протоколы транслятора

|  |  |
| --- | --- |
| Протокол | Описание |
| Файл журнала (log) | Файл с протоколом работы транслятора языка WSA-2020. Содержит таблицы лексем и идентификаторов, протокол работы синтаксического анализатора и дерево разбора, а также результат преобразования выражений к польской обратной записи. |
| Выходной файл (out) | Содержит результат работы программы – код на языке ассемблера. |

# 3.Разработка лексического анализатора

## 3.1 Структура лексического анализатора

Лексический анализатор – программа реализующая этап лексического анализа в работе транслятора. Данная программа преобразует исходный текст, заменяя лексические единицы языка их внутренним представлением – лексемами. Для описания лексики языка программирования применяются регулярные грамматики, относящиеся к типу 3 иерархии Хомского. Язык, заданный регулярной грамматикой, называется регулярным языком. Регулярный язык однозначно задается регулярным выражением, а распознавателями для регулярных языков являются конечные автоматы.

Функции лексического анализатора:

* удаление “пустых” символов и пробелов. К “пустым” символом относятся: пробелы, табуляции, переходы в новую строку;
* распознавание идентификаторов и ключевых слов;
* распознавание констант;
* распознавание разделителей и символов-операций;

Структура лексического анализатора представлена на рисунке 3.1.

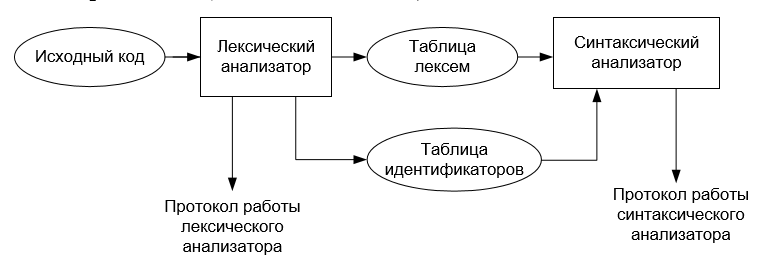


Рисунок 3.1. Структура лексического анализатора

## 3.2 Контроль входных символов

Таблица контроля входных символов представлена на рисунке 3.2.

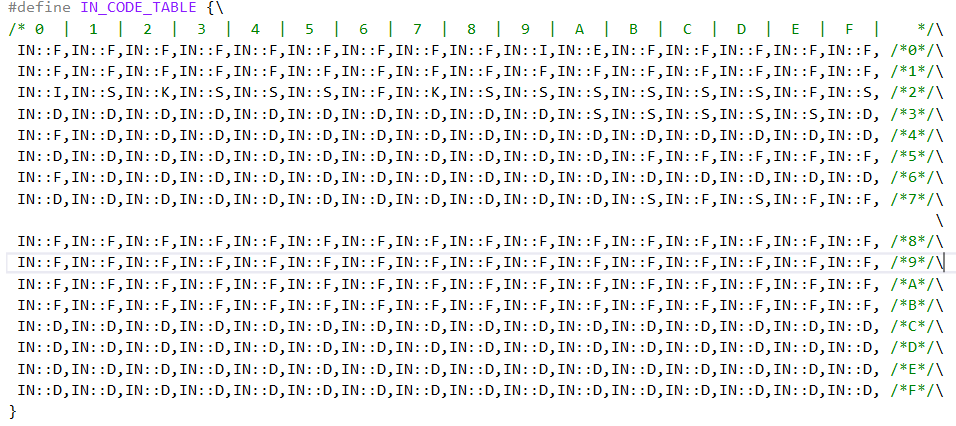


Рисунок 3.2. Таблица контроля входных символов

Соответствие символов и их значений описаны в таблице 3.1.

Таблица 3.1.Соответствие символов и их значений в таблице

|  |  |
| --- | --- |
| Символ | Значение |
| F | Запрещённый символ |
| I | Игнорируемый символ |
| E | Символ окончания строки |
| S | Разрешённый символ, не входящий в слово |
| D | Разрешённый символ, входящий в слово |
| K | Одинарная или двойная кавычка |

## 3.3 Удаление избыточных символов

Перечень избыточных символов: пробел, табуляция, переход в новую строку. Удаление избыточных символов происходит на этапе разбиения исходного текста на токены.

Описание алгоритма удаления избыточных символов:

1. Считываем входной поток символов:
2. При встрече символа-разделителя или избыточного символа, записываем считаное слово:
3. Если символ избыточный мы не записываем его в слово, а игнорируем.

## 3.4 Перечень ключевых слов

Соответствие токенов и лексем приведено в таблице 3.2.

Таблица 3.2 Соответствие токенов и лексем

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Описание |
| short, string, bool, char | t | Типы данных языка. |
| Идентификатор | i | Максимальный размер -8 символов. |
| Литерал | l | Литерал соответствующий доступному типу. |
| func | f | Объявление функции. |
| proc | p | Объявление процедуры. |
| var | v | Объявление переменной. |
| return | r | Возврат значения и выход из функции. |
| extr | e | Подключение внешней функции . |
| write | w | Вывод в консоль. |
| while | c | Оператор цикла. |
| main | m | Точка входа в программу. |
| +,-,\*,/,%,<,>,==,!= | o | Знаки операций. |
| $ | $ | Вызов функции. |
| ; | ; | Разделитель выражений. |
| , | , | Разделитель параметров функции или процедуры. |
| { | } | Начало программного блока. |
| } | } | Конец программного блока. |
| ( | ( | Приоритет операций, передача параметров функции или процедуры. |
| ) | ) | Окончание блока приоритета операций или передачи параметров. |
| = | = | Оператор присваивания. |

Каждому выражению соответствует детерминированный конечный автомат. Автоматы собраны в массив и туда передаются токены, если есть совпадение по первому символу, то токен проверяется по соответствующему графу переходов. Если на этапе прохода по массиву токен не был разобран проверяем его на идентификатор или операцию, при помощи соответствующих конечных автоматов. В случае успешного разбора символ записывается в таблицу лексем. Если символ является идентификатором или литералом, то он заносится и в таблицу идентификаторов. Структура конечного автомата приведена на рисунке 3.3, пример графов переходов на рисунке 3.4.

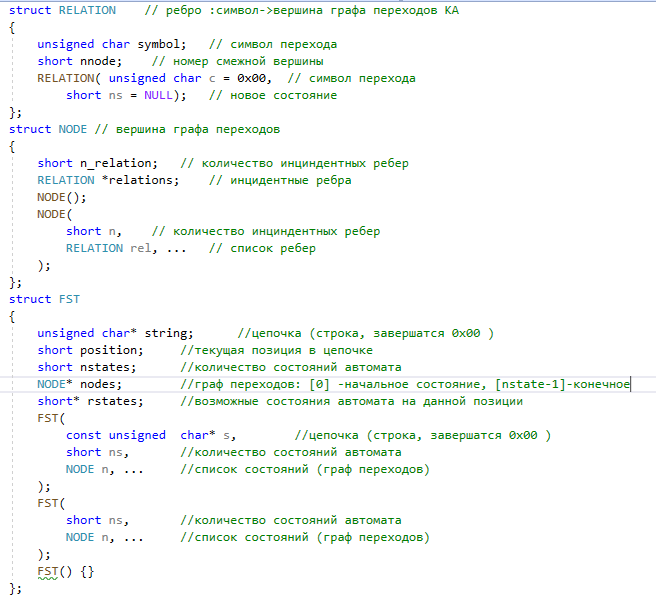


Рисунок 3.3 Структура конечного автомата

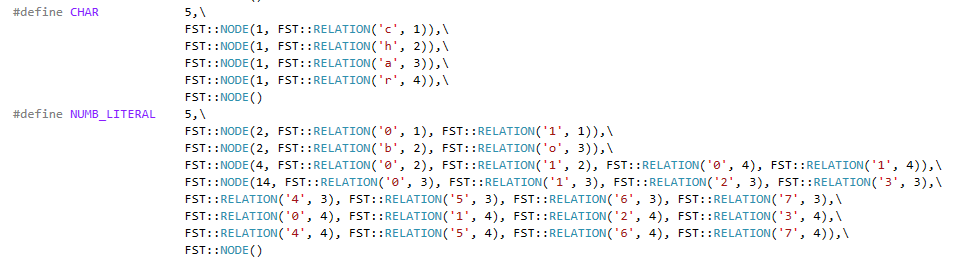


Рисунок 3.4 Пример графов переходов

## 3.5 Основные структуры данных

Основными структурами данных лексического анализатора являются таблица лексем и таблица идентификаторов. Таблица лексем содержит номер лексемы, лексему, полученную при разборе, номер строки в исходном коде, и номер в таблице идентификаторов, если лексема является идентификаторов. Таблица идентификаторов содержит имя идентификатора, номер в таблице лексем, тип данных, тип идентификатора и значение. Код структуры таблицы лексем на языке С++ представлен на рисунке 3.5, а код структуры таблицы идентификаторов на рисунке 3.6.

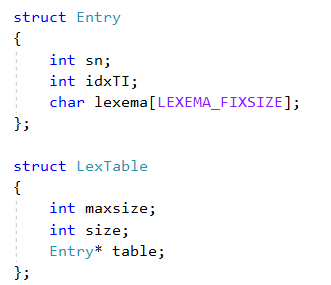


Рисунок 3.5 Структура таблицы лексем

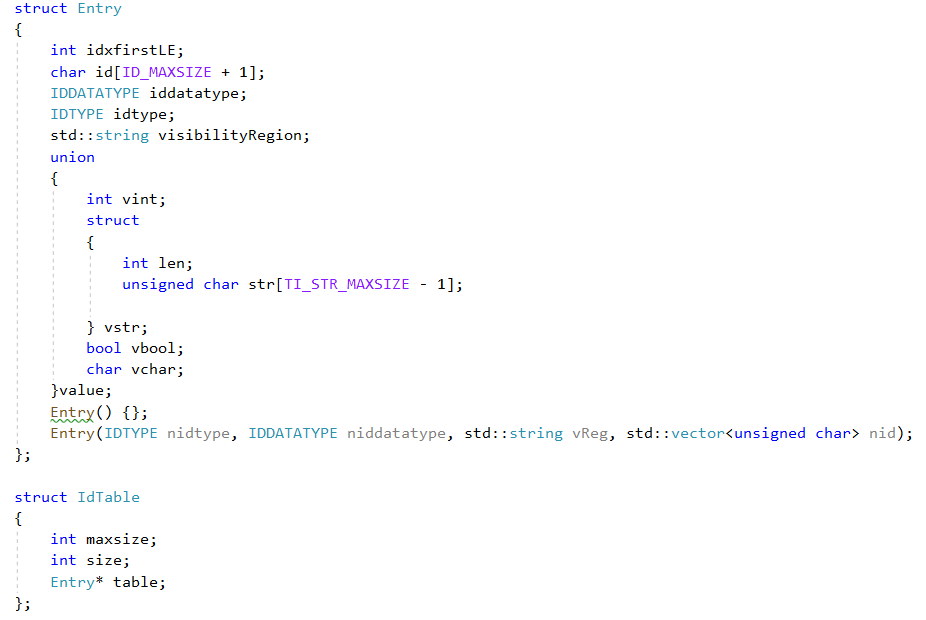


Рисунок 3.6 Структура таблицы идентификаторов

## 3.6 Структура и перечень сообщений лексического анализатора

Структура сообщений содержит информацию о номере сообщения, номер строки и позицию начала токена, где было вызвано сообщение в исходном коде, информацию об ошибке. Перечень сообщений представлен на рисунке 3.7.

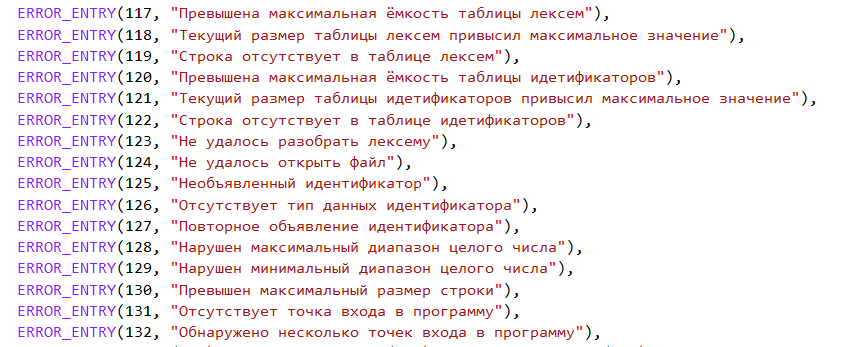


Рисунок 3.7 Перечень сообщений лексического анализатора

## 3.7 Принцип обработки ошибок

Ошибки, возникающие в процессе трансляции программы, фиксируются в протокол, заданный входным параметрами. В случае возникновения ошибок транслятор прекращает свою работу.

## 3.8 Параметры лексического анализатора

Результаты работы лексического анализатора, а именно таблицы лексем и идентификаторов выводятся в файл журнала.

## 3.9 Алгоритм лексического анализа

Описания алгоритма:

1. Проверяем входной поток символов на допустимость, отбрасываем не нежные символы, по символам-сепараторам выделяем слова и добавляем в массив токенов.
2. Токены из массива поочередно подаются на функцию распознавания лексемы.
3. При успешном разборе информация заносится в таблицу лексем и идентификаторов, далее алгоритм возвращается ко второму этапу.
4. При неуспешном разборе выдаётся сообщение об ошибке.
5. Формируется протокол работы.

Распознавание цепочек основывается на работе конечных автоматов. Работу конечного автомата можно проиллюстрировать с помощью графа переходов. Пример графа для цепочки «char» представлен на рисунке 3.8, где S0 – начальное, а S4 – конечное состояние автомата.

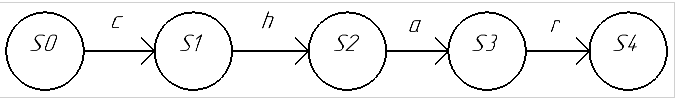


Рисунок 3.8 Граф переходов для цепочки “char”

## 3.10 Контрольный пример

Результат работы лексического анализатора в виде таблиц лексем и идентификаторов, соответствующих контрольному примеру, представлен в приложении Б.

# 4.Разработка синтаксического анализатора

## 4.1 Структура синтаксического анализатора

Синтаксический анализатор: составная часть транслятора, выполняющая синтаксический анализ, то есть проверяет исходный код на соответствие правилам грамматики. Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов. Выходная информация – дерево разбора. Описание структуры синтаксического анализатора языка представлено на рисунке 4.1.

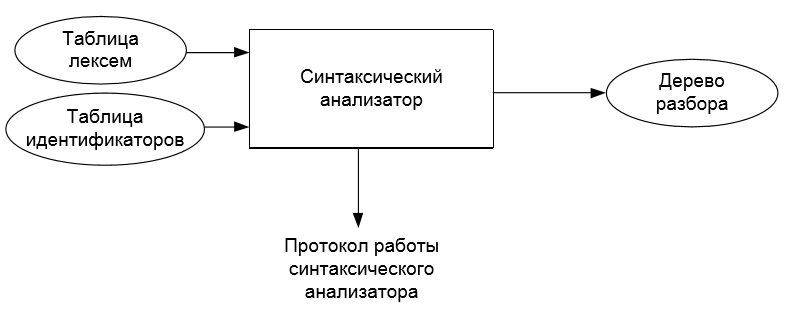


Рисунок 4.1 Структура синтаксического анализатора

## 4.2 Контекстно-свободная грамматика, описывающая синтаксис языка

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

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

N – множество нетерминальных символов,

P – множество правил языка ,

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.

Описание нетерминальных символов содержится в таблице 4.1.

Таблица 4.1 Описание нетерминальных символов

|  |  |  |
| --- | --- | --- |
| Нетерминал | Правило | Описание |
| S | S->m{N}  S->m{N}S  S->fti(F){NrE;}S  S->fti(F){NrE;}  S->fti(){NrE;}S  S->fti(){NrE;}  S->pi(F){N}S  S->pi(F){N}  S->pi(){N}S  S->pi(){N}S  S->eti(F);  S->eti(F);S | Стартовый символ, описывает общую структуру программы. |
| N | N->vti;  N->rE;  N->i=E;  N->vti;N  N->rE;N  N->i=E;N  N->wE;  N->wE;N  N->c(E){N}  N->c(E){N}  N->$i(W);N  N->$i();N  N->$i();  N->$i(W); | Основные инструкции языка |
| E | E->i  E->l  E->(E)  E->$i(W)  E->$i()  E->iM  E->lM  E->(E)M  E->$i(W)M  E->$i()M | Выражение |
| M | M->oE  M->oEM | Арифметическая или логическая операция |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Нетерминал | Правило | Описание |
| F | F->ti  F->ti,F | Описание параметров функции или процедуры |
| W | W->i  W->l  W->i,W  W->l.W | Передача параметров функции или процедуры |

## 4.3 Построение конечного магазинного автомата

Конечный автомат с магазинной памятью представляет собой семерку. Описание компонентов магазинного автомата представлено в таблице 4.2.

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 представляет из себя множества терминальных и нетерминальных символов. |
|  | Алфавит специальных магазинных символов | Алфавит магазинных символов содержит стартовый символ и маркер дна стека (представляет из себя символ $) |
|  | 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 |
|  | Начальное состояние магазина автомата | Символ маркера дна стека $ |

Продолжение таблицы 4.2

|  |  |  |
| --- | --- | --- |
| Компонента | Определение | Описание |
|  | Множество конечных состояний | Конечные состояние заставляют автомат прекратить свою работу. Конечным состоянием является пустой магазин автомата и совпадение позиции на входной ленте автомата с размером ленты |

## 4.4 Основные структуры данных

Основные структуры данных синтаксического анализатора представляются в виде структуры магазинного конечного автомата, выполняющего разбор исходной ленты, и структуры грамматики Грейбах, описывающей синтаксические правила языка WSA-2020. Правила языка WSA-2020 в приложении В.

## 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. Иначе генерируется исключение.

## 4.6 Структура и перечень сообщений синтаксического анализатора

Перечень сообщений синтаксического анализатора представлен на рисунке 4.2.

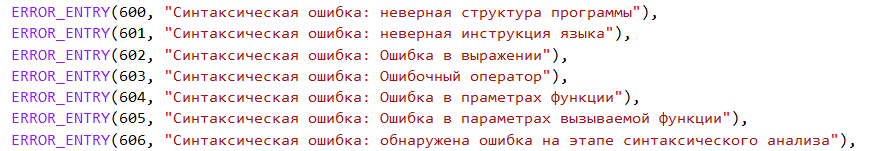


Рисунок 4.2 Сообщения синтаксического анализатора

## 4.7. Параметры синтаксического анализатора и режимы его работы

Входной информацией для синтаксического анализатора является таблица лексем и идентификаторов. Так же используется описание грамматики в форме Грейбах. Результаты работы синтаксического разбора, а именно дерево разбора и протокол работы автомата с магазинной памятью выводятся в журнал работы программы.

## 4.8. Принцип обработки ошибок

Синтаксический анализатор выполняет разбор исходной последовательности лексем до тех пор, пока не дойдёт до конца цепочки лексем или не найдёт ошибку. Тогда анализ останавливается и выводится сообщение об ошибке (если она найдена). Если в процессе анализа находятся более трёх ошибок, то анализ останавливается.

## 4.9. Контрольный пример

Результаты работы синтаксического разбора, а именно дерево разбора и протокол работы автомата с магазинной памятью приведены в приложении В.

# 5. Разработка семантического анализатора

## 5.1 Структура семантического анализатора

Семантический анализатор принимает на свой вход результаты работ лексического и синтаксического анализаторов, то есть таблицы лексем, идентификаторов и дерево разбора, и последовательно ищет необходимые ошибки. Некоторые проверки осуществляются в процессе лексического анализа. Общая структура обособленно работающего (не параллельно с лексическим анализом) семантического анализатора представлена на рисунке 5.1.



Рисунок 5.1. Структура семантического анализатора

## 5.2 Функции семантического анализатора

Семантический анализатор выполняет проверку на основные правил 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.

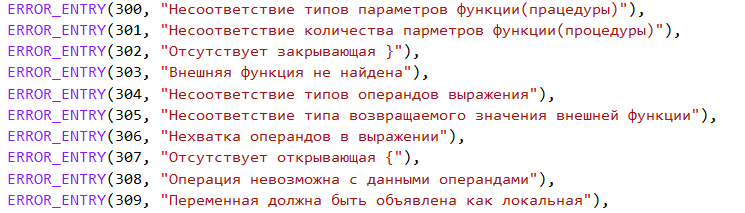


Рисунок 5.2 – Перечень сообщений семантического анализатора

## 5.4 Принцип обработки ошибок

Ошибки, возникающие в процессе трансляции программы, фиксируются в протокол, заданный входным параметрами. В случае возникновения ошибок происходит их протоколирование с номером ошибки и диагностическим сообщением, а также остановка работы транслятора.

## 5.5 Контрольный пример

Соответствие примеров некоторых ошибок в исходном коде и диагностических сообщений об ошибках приведено в таблице 5.1.

Таблица 5.1. Примеры диагностики ошибок

|  |  |
| --- | --- |
| Исходный код | Текст сообщения |
| main  {  var short i;  i=0b11; | Ошибка 302: Отсутствует закрывающая }(строка:4 позиция:0) |
| main  {  var short i;  var string s;  var bool b;  b =s+i; } | Ошибка 304: Несоответствие типов операндов выражения (строка:5 позиция:0) |
| main  {  var short s1;  var string s2;  s1 = "Число";  s2 = "Строка";  s1 = s1-s2;  } | Ошибка 308: Операция невозможна с данными операндами (строка:6 позиция:0) |

# 6. Вычисление выражений

## 6.1 Выражения, допускаемые языком

В языке WSA-2020 допускаются вычисления выражений целочисленного типа данных, а также конкатенация строк, с поддержкой вызова функций внутри выражений. Приоритет арифметических операций представлен на таблице 6.1.

Таблица 6.1. Приоритеты арифметических операций

|  |  |
| --- | --- |
| Операция | Значение приоритета |
| ( ) | 0 |
| , | 1 |
| \* | 3 |
| / | 3 |
| + | 2 |
| - | 2 |
| {} | 4 |

## 

## 6.2 Польская запись и принцип её построения

Все выражения языка WSA-2020 преобразовываются к обратной польской записи.

Польская запись - это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок. Отличие их от классического, инфиксного способа заключается в том, что знаки операций пишутся не между, а до или после аргументов.

Алгоритм построения польской записи:

1. исходная строка: выражение;
2. результирующая строка: польская запись;
3. стек: пустой; - исходная строка просматривается слева направо;
4. операнды переносятся в результирующую строку в порядке их следования;
5. операция записывается в стек, если стек пуст или в вершине стека лежит отрывающая скобка;
6. операция выталкивает все операции с большим или равным приоритетом в результирующую строку;
7. запятая не помещается в стек, и если в стеке есть операции, то все выбираются в строку;
8. открывающая скобка помещается в стек; - закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются;
9. фигурная открывающая скобка помещается в стек;
10. фигурная закрывающая скобка выталкивает все до открывающей фигурной скобки и генерирует последовательность @n (индекс n указывает число операндов, разделенных запятыми);
11. по концу разбора исходной строки все операции, оставшиеся в стеке, выталкиваются в результирующую строку.

## 6.3 Программная реализация обработки выражений

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

## 6.4 Контрольный пример

Пример преобразования выражений из контрольных примеров к обратной польской записи представлен в таблице 6.2. В приложении Г приведены изменённые таблицы лексем и идентификаторов, отображающие результаты преобразования выражений в польский формат.

Таблица 6.2. Преобразование выражений к ПОЛИЗ

|  |  |
| --- | --- |
| Выражение | Обратная польская запись для выражения |
| i=$i(l,l)o{+}l; | i=$ill@2lo{+} ; |
| $i(i,i); | $iii@2 ; |
| i=io{%}i; | i=iio{%}; |

# 7. Генерация кода

## 7.1 Структура генератора кода

В языке WSA-2020 генерация кода является заключительным этапом трансляции. Генератор принимает на вход таблицы лексем и идентификаторов, полученные в результате лексического анализа. В соответствии с таблицей лексем строится выходной файл на языке ассемблера. В случае возникновения ошибок генерация кода не будет осуществляться. Структура генератора кода WSA-2020 представлена на рисунке 7.1.

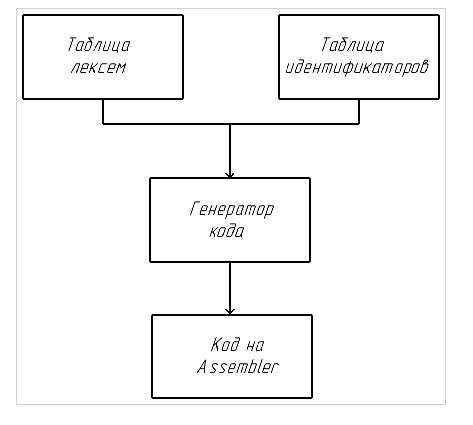


Рисунок 7.1 Структура генератора кода

## 7.2 Представление типов данных в оперативной памяти

Элементы таблицы идентификаторов расположены в сегментах .data и .const языка ассемблера. Соответствия между типами данных идентификаторов на языке WSA-2020 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 Соответствия типов идентификаторов языка WSA-2020 и языка ассемблера

|  |  |  |
| --- | --- | --- |
| Тип на языке WSA-2020 | Тип на языке ассемблера | Описание |
| short | sword | Хранит целочисленный тип данных. |
| string, char | byte | Хранит указатель на начало строки. Строка должна завешаться нулевым символом. |
| bool | word | Хранит булевский тип данных. Может принимать значения 0 и 1. |

## 7.3 Статическая библиотека

В языке WSA-2020 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++. Объявление функций статической библиотеки осуществляется с помощью инструкции языка: extr <тип данных><идентификатор>(<параметр>,…), либо автоматически генерируется на языке ассемблера. Стандартная библиотека описана в таблице 7.2.

Таблица 7.2 Функции статической библиотеки

|  |  |  |
| --- | --- | --- |
| Функция | Тип подключения | Описание |
| int random(short a, short b) | В исходном коде | Генерирует случайное целое число в диапазоне от b до a. |
| int len(short l, char\* str) | В исходном коде | Возвращает длину строки. |
| char\* concat(short l1, char\* str1, short l2, char\* str2) | Автоматически | Невозможно вызвать в исходном коде. Вызывается в реализации операции + для строковых типов данных и производит конкатенацию строк. |
| bool writeNumberBin(short number) | Автоматически | Невозможно вызвать в исходном коде. Вызывается в реализации инструкции write для вывода числа в двоичном формате. |
| bool writeOct(short number) | В исходном коде | Выводит число в восьмеричном формате. |
| bool writeStr(short l, char\* str) | Автоматически | Невозможно вызвать в исходном коде. Вызывается в реализации инструкции write для вывода строки либо символа. |
| bool writeBool(int b) | Автоматически | Невозможно вызвать в исходном коде. Вызывается в реализации инструкции write для вывода булевского типа данных. |
| short stringEquel(short l1, char\* str1, short l2, char\* str2) | Автоматически | Невозможно вызвать в исходном коде. Вызывается в реализации операций <,>,==,!= для строк. |

## 

## 7.4 Особенности алгоритма генерации кода

В языке WSA-2020 генерация кода строится на основе таблиц лексем и идентификаторов и на выходе получаем asm-файл.

## 7.5 Входные параметры генератора кода

На вход генератор кода получает результат работы лексического анализа, в котором выражения преобразованы к ПОЛИЗ.

## 7.6 Контрольный пример

Результат генерации ассемблерного кода на основе контрольного примера (приложение А), приведен в приложении Д. Результат работы контрольного примера, приведён на рисунке 7.2.

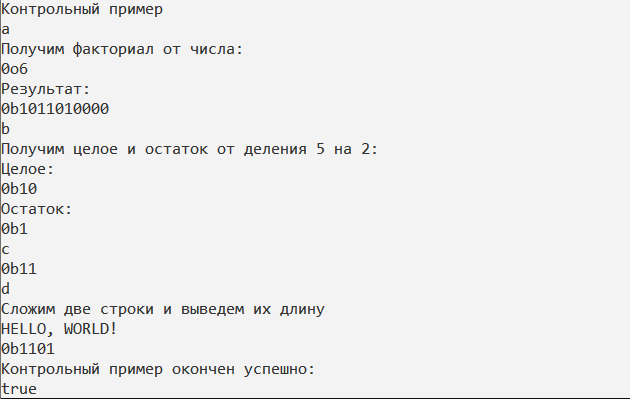


Рисунок 7.2 Результат работы программы на языке ВАА-2019

# 8. Тестирование транслятора

## 8.1 Общие положения

При возникновении ошибки на каком-либо этапе трансляции, она обрабатывается в главном файле программы: ошибка выводится на консоль и записывается в файл журнала.

## 8.2 Результаты тестирования

Результаты тестирования приведены в таблице 8.1

Таблица 8.1 Результаты тестирования транслятора

|  |  |
| --- | --- |
| Исходный код | Сообщение |
| &main | Ошибка 111: Недопустимый символ в исходном файле(-in) |
| main  {  var short a;  a = b; } | Ошибка 125: Необъявленный идентификатор |
| func short fs()  {  ret 0b11; } | Ошибка 131: Отсутствует точка входа в программу |
| main  {  short i;  i=3;  } | Ошибка 125: Необъявленный идентификатор |
| main  {  var string str;  var string str; } | Ошибка 127: Повторное объявление идентификатора |
| func short fs(short i)  {  i = i + i;  ret i;  }  main  {  var short a;  a = $fs(0b1,0b1,0b1);  } | Ошибка 301: Несоответствие количества параметров функции (процедуры) |

Продолжение таблицы 8.1

|  |  |
| --- | --- |
| Исходный код | Сообщение |
| var short i;  main  {  write "Hello";  } | Ошибка 309: Переменная должна быть объявлена как локальная |
| main  {  var short a;  a = 0b1 + "Hello";  } | Ошибка 304: Несоответствие типов операндов выражения |
| main ret 0b1; | Ошибка 600: Синтаксическая ошибка: неверная структура программы |
| main  {  var short i;  i = 0b1;  i=i++i;  } | Ошибка 602: Синтаксическая ошибка: Ошибка в выражении |
| func short fs(short i)  {  i = i + i;  ret i;  }  main  {  var short a;  a = $fs("Число");  } | Ошибка 300: Несоответствие типов параметров функции (процедуры) |

# 

# Заключение

В ходе выполнения курсовой работы был разработан транслятор для языка программирования WSA-2020. Были выполнены основные задачи данной курсовой работы:

* Сформулирована спецификация языка WSA-2020;
* Разработан лексический анализатор;
* Разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика;
* Разработан синтаксический анализатор;
* Разработан семантический анализатор;
* Разработан транслятор кода на язык ассемблера;
* Проведено тестирование транслятора.

Окончательная версия языка WSA-2020 включает:

* 4 типа данных;
* Поддержка оператора вывода;
* Возможность вызова функций стандартной библиотеки;
* Наличие 5 арифметических операторов для вычисления выражений;
* Поддержка функций, процедур и оператора цикла;
* Структурированная и классифицированная система для обработки ошибок пользователя.

Проделанная работа позволила получить необходимое представление о структурах и процессах, использующихся при построении трансляторов, а также основные различия и преимущества тех или иных средств трансляции.

# Список использованных источников

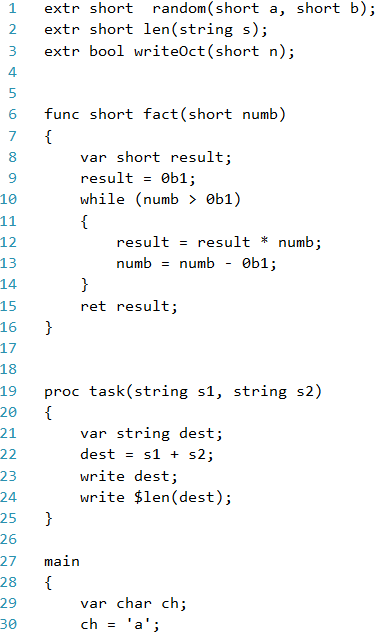
1. Ахо, А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003. – 768с.

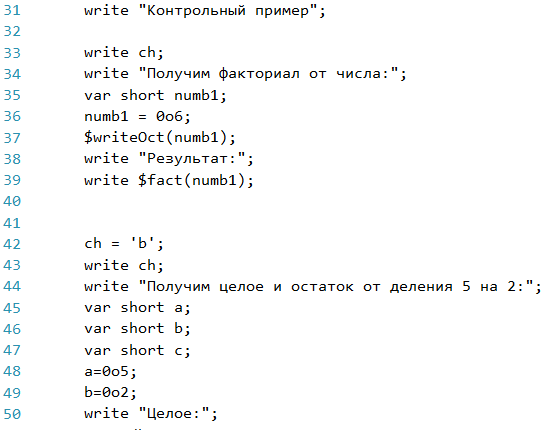
2. Ахо, А. Теория синтаксического анализа, перевода и компиляции /А. Ахо, Дж. Ульман. – Москва : Мир, 1998. – Т. 2 : Компиляция. - 487 с.

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

# Приложение А

Исходный код программы на языке WSA-2020





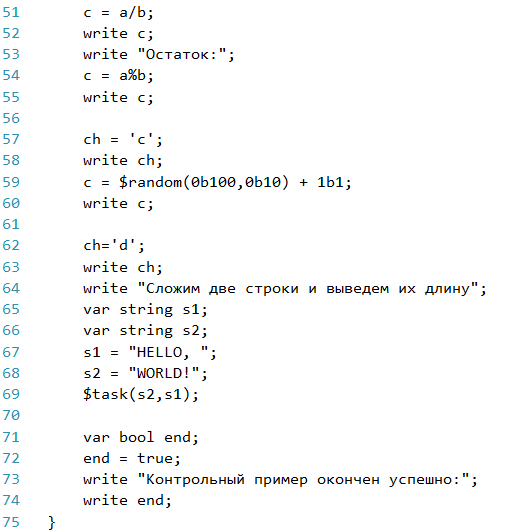
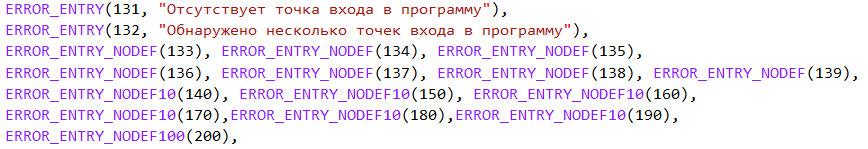
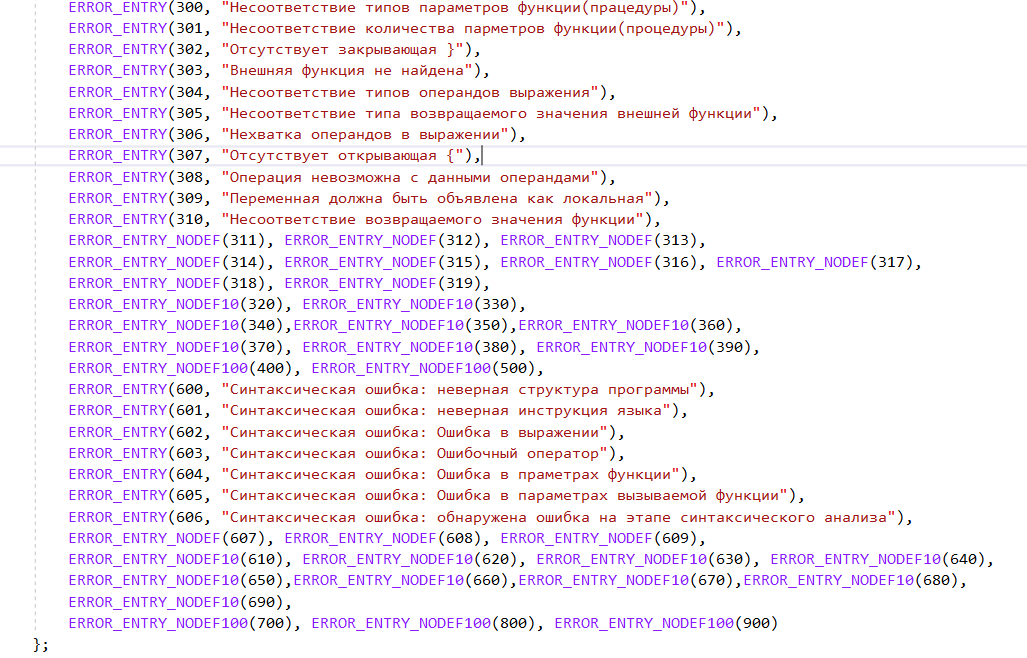


Таблица ошибок языка WSA-2020

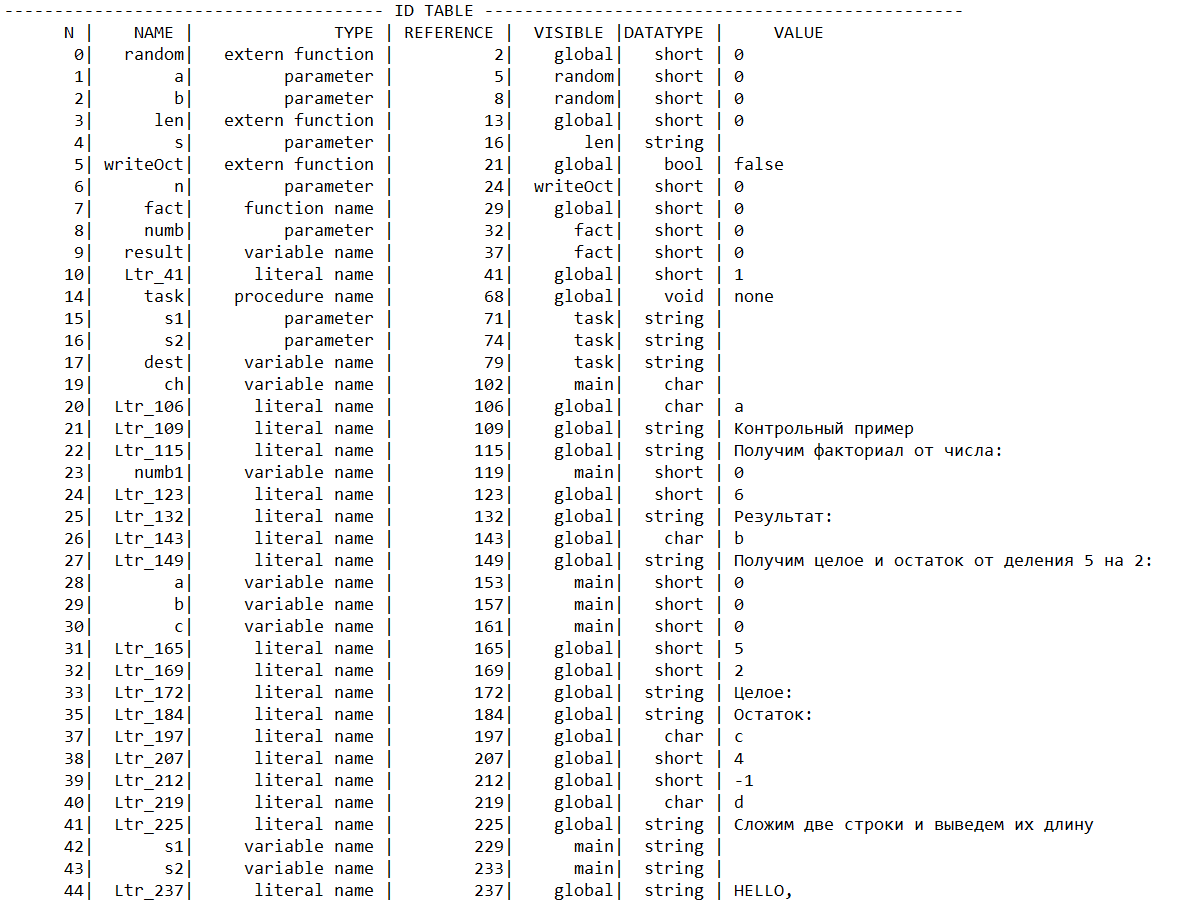


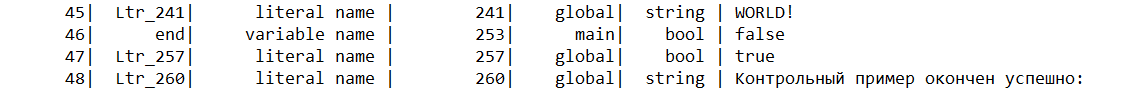




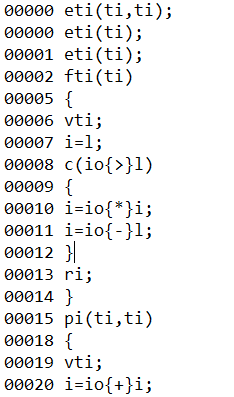
## Приложение Б

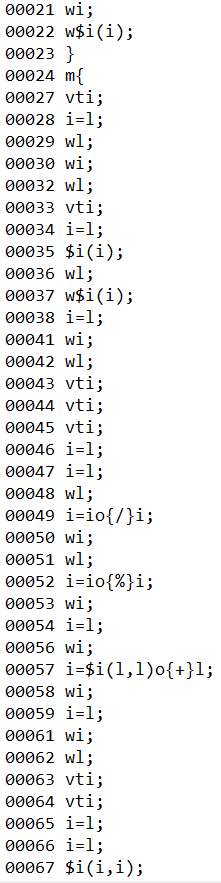
Таблица идентификаторов контрольного примера

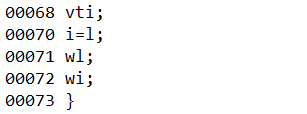




Лексемы соответствующие контрольному примеру

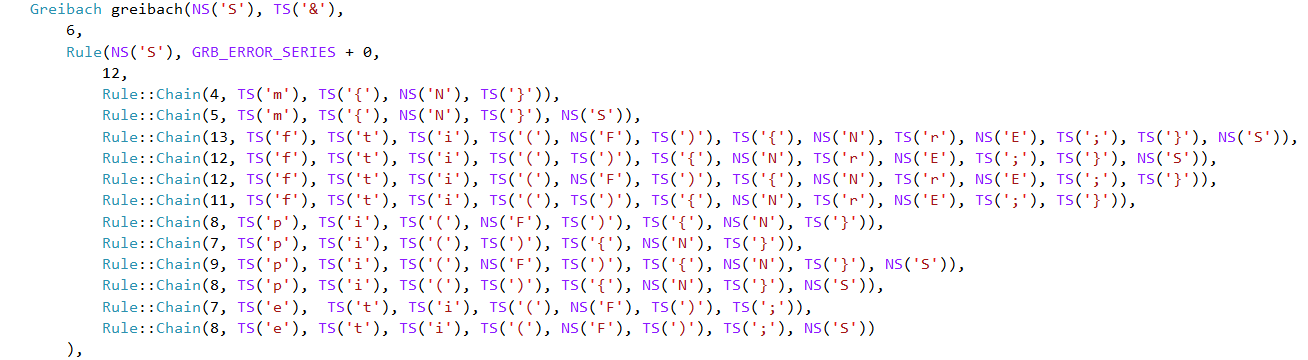


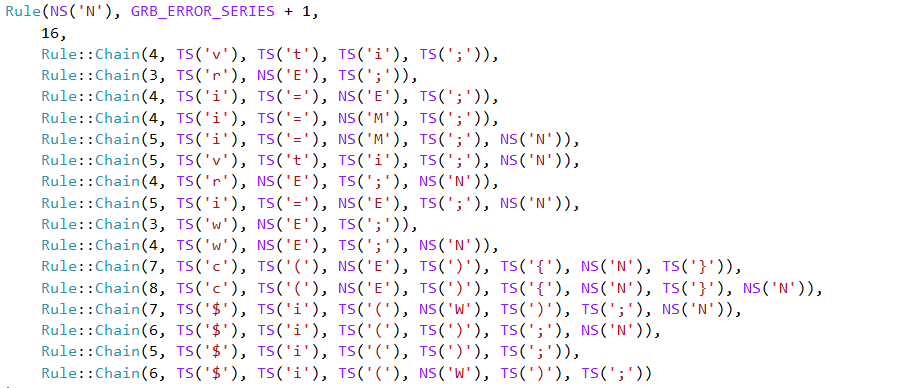


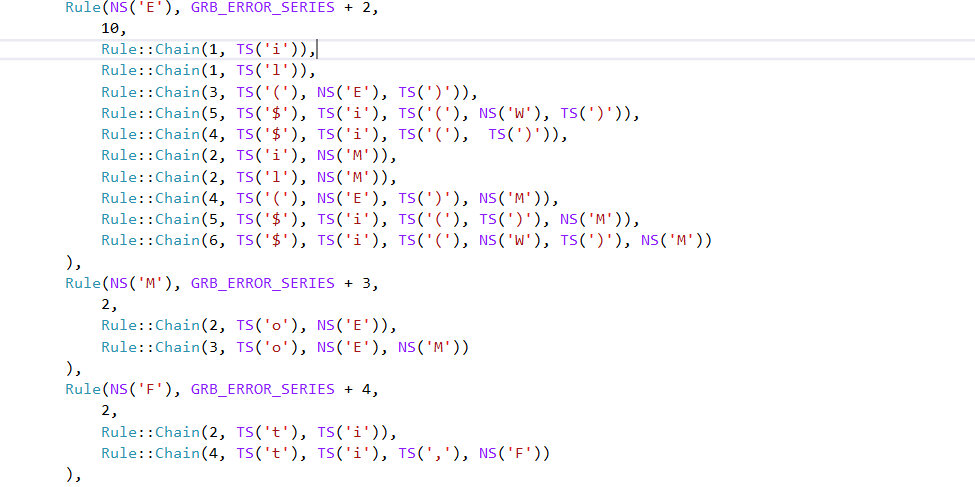


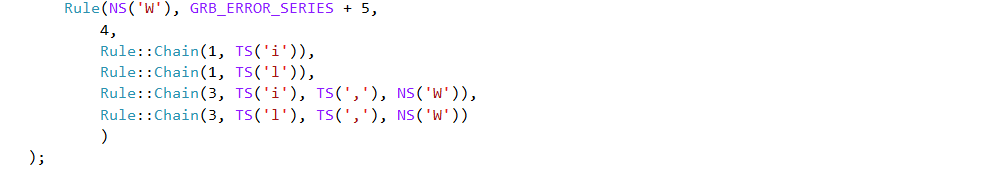
# Приложение В

Грамматика языка

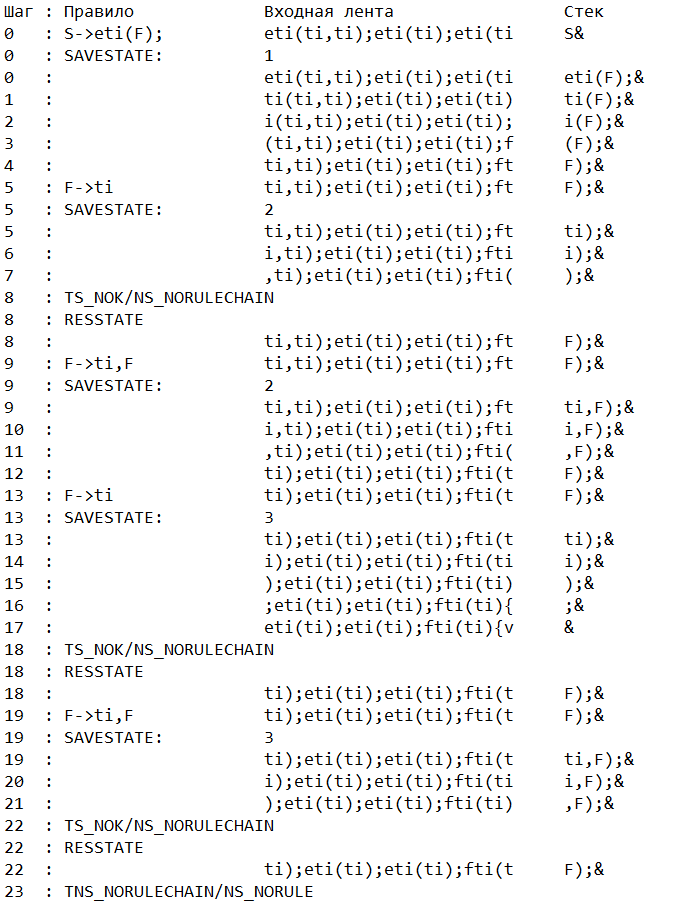


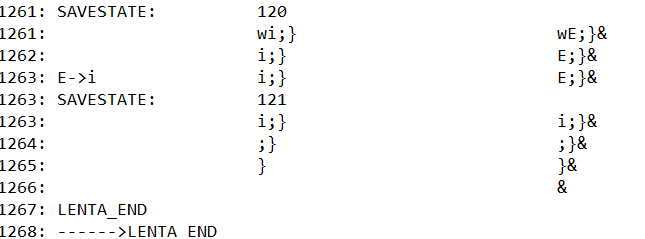




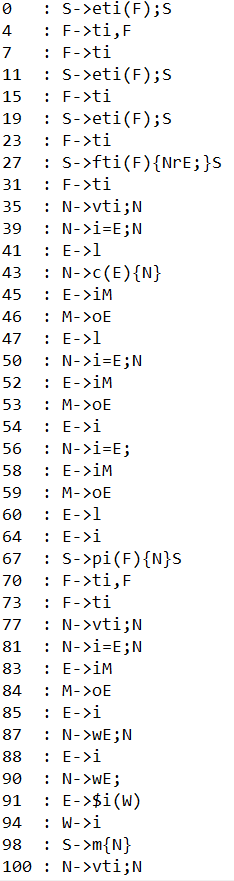


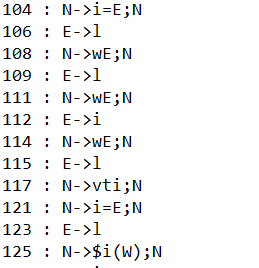
Разбор исходного кода синтаксическим анализатором

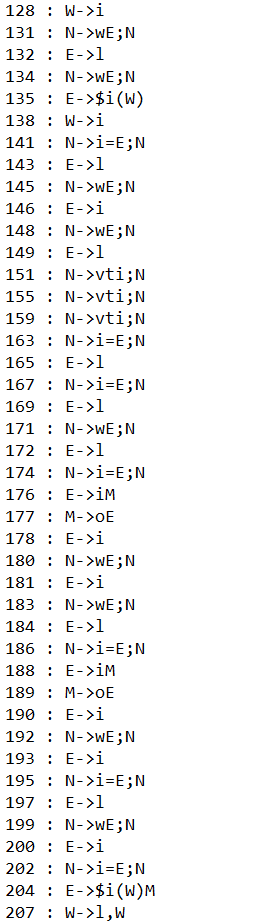


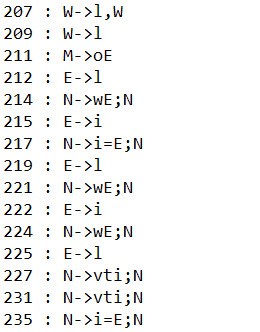


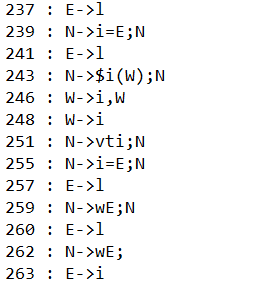
Дерево разбора





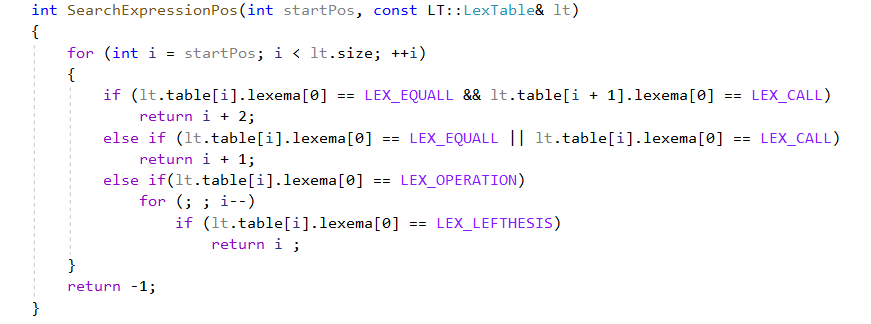


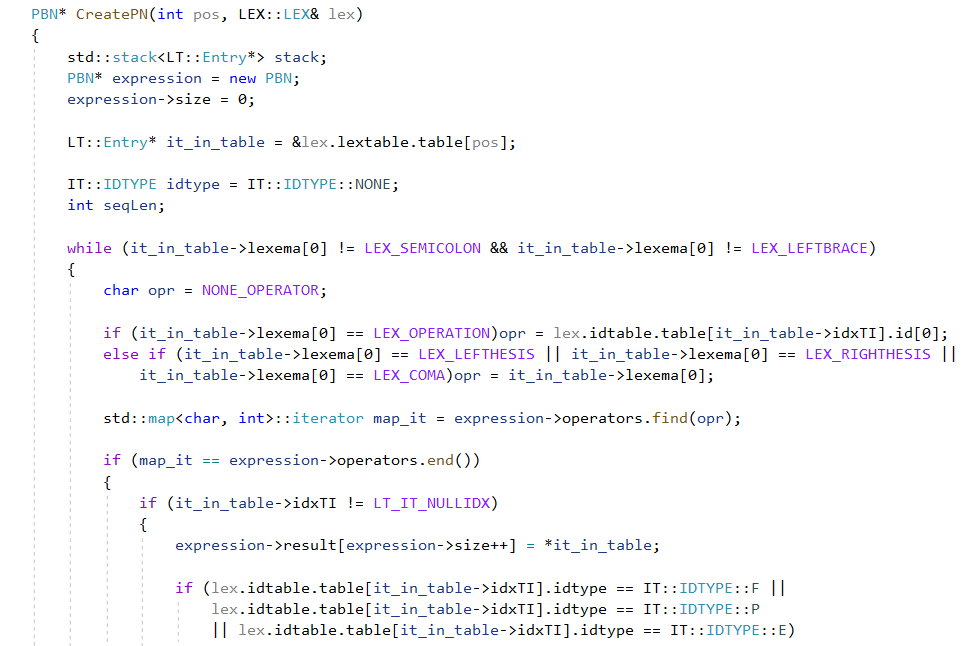


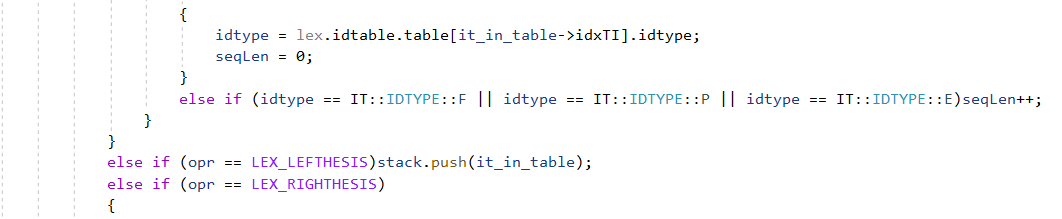


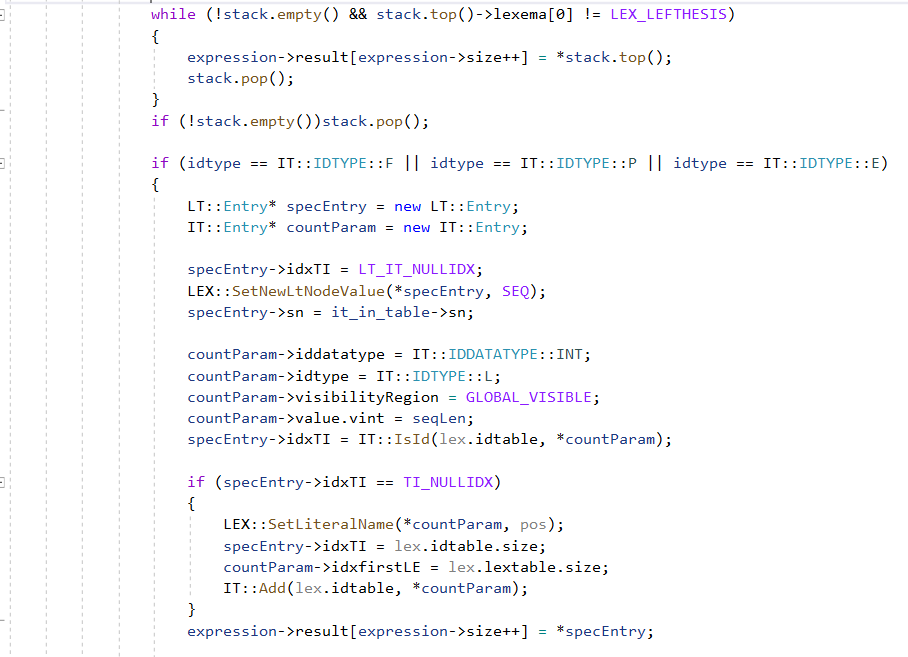
# Приложение Г

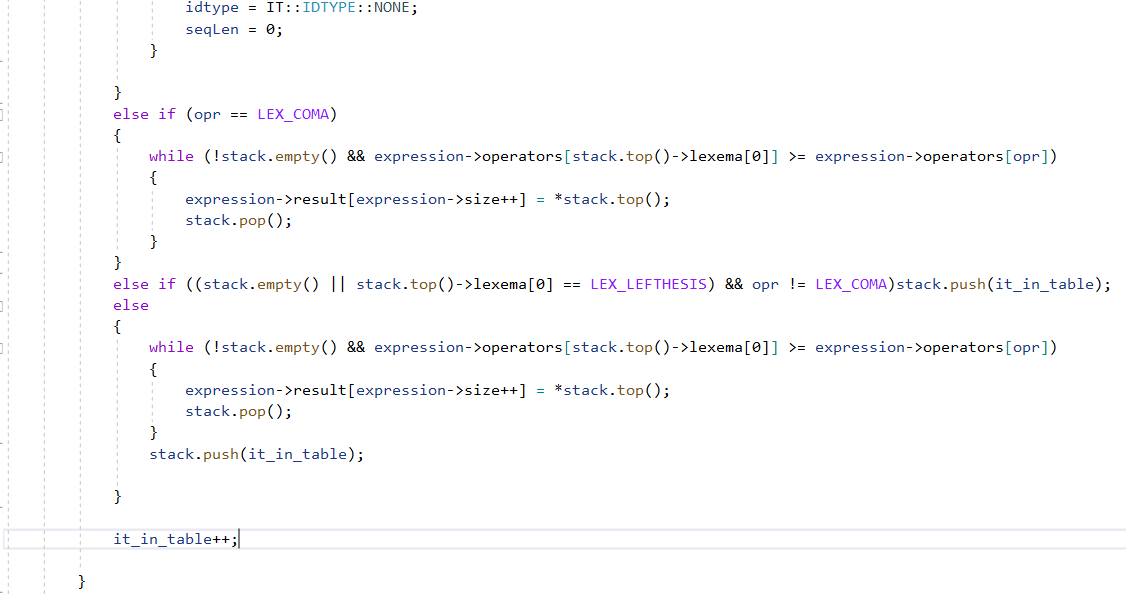
Программная реализация механизма преобразования в ПОЛИЗ

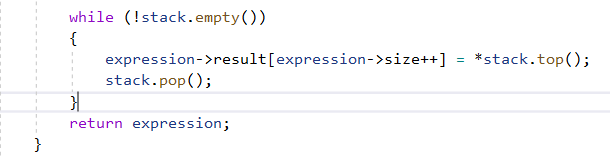


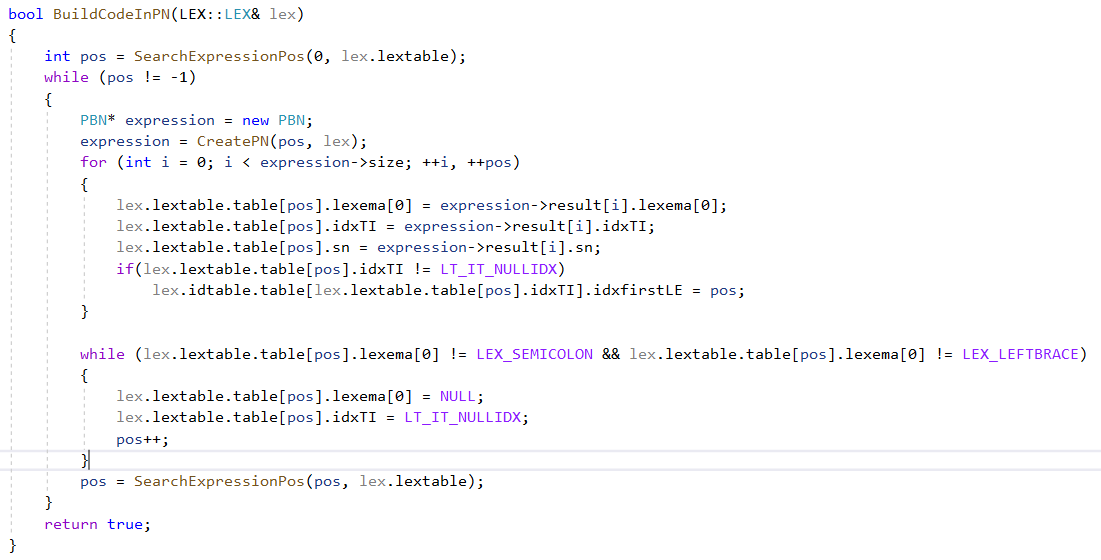




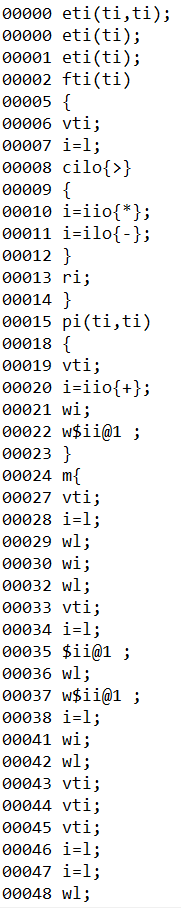


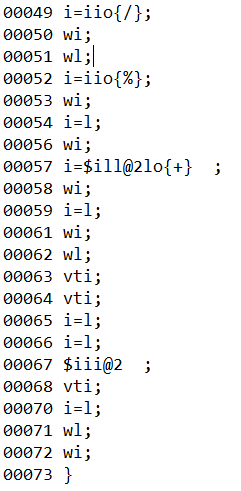






Лексемы соответствующие контрольному примеру после преобразования выражений в ПОЛИЗ





# Приложение Д

Результат генерации кода исходного примера в язык ассемблера.

