МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационныx

технологий

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора SDE-2019»

Выполнил студент Селицкий Данил Евгеньевич

(Ф.И.О.)

Руководитель проекта ст.пр. Наркевич Аделина Сергеевна

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

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

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

Консультанты ст.пр. Наркевич Аделина Сергеевна

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

Нормоконтролер ст.пр. Наркевич Аделина Сергеевна

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

Курсовой проект защищен с оценкой

Минск 2019

**Содержание**

[1. Спецификация языка программирования 6](#_Toc27686413)

[**1.1. Характеристика языка программирования** 6](#_Toc27686414)

[**1.2. Алфавит языка** 6](#_Toc27686415)

[**1.3. Применяемые сепараторы** 7](#_Toc27686416)

[**1.5. Типы данных** 7](#_Toc27686417)

[**1.6. Преобразование типов данных** 8](#_Toc27686418)

[**1.7. Идентификаторы** 8](#_Toc27686419)

[**1.8. Литералы** 8](#_Toc27686420)

[**1.9. Объявление данных и область видимости** 8](#_Toc27686421)

[**1.10. Инициализация данных** 9](#_Toc27686422)

[**1.11 Инструкции языка** 9](#_Toc27686423)

[**1.12 Операции языка** 10](#_Toc27686424)

[**1.13 Выражения и их вычисление** 10](#_Toc27686425)

[**1.14 Программные конструкции языка** 10](#_Toc27686426)

[**1.15 Область видимости идентификаторов** 11](#_Toc27686427)

[**1.16 Семантические проверки** 12](#_Toc27686428)

[**1.17 Распределение оперативной памяти на этапе выполнения** 12](#_Toc27686429)

[**1.18 Стандартная библиотека и её состав** 12](#_Toc27686430)

[**1.19 Ввод и вывод данных** 13](#_Toc27686431)

[**1.20 Точка** **входа** 13](#_Toc27686432)

[**1.21 Препроцессор** 13](#_Toc27686433)

[**1.22 Соглашения о вызовах** 13](#_Toc27686434)

[**1.23 Объектный код** 13](#_Toc27686435)

[**1.24 Классификация сообщений транслятора** 13](#_Toc27686436)

[**1.25 Контрольный пример** 14](#_Toc27686437)

[2. Структура транслятора 16](#_Toc27686438)

[**2.1 Компоненты транслятора, их назначение и принципы взаимодействия** 16](#_Toc27686439)

[**2.2 Перечень входных параметров транслятора** 17](#_Toc27686440)

[**2.3 Перечень протоколов, формируемых транслятором и их содержимое** 17](#_Toc27686441)

[**3.1 Структура лексического анализатора** 19](#_Toc27686442)

[**3.2 Контроль входных символов** 19](#_Toc27686443)

[**3.3 Удаление избыточных символов** 20](#_Toc27686444)

[**3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов** 20](#_Toc27686445)

[**3.5 Основные структуры данных** 21](#_Toc27686446)

[**3.6 Структура и перечень сообщений лексического анализатора** 21](#_Toc27686447)

[**3.7 Принцип обработки ошибок** 22](#_Toc27686448)

[**3.8 Параметры лексического анализатора** 22](#_Toc27686449)

[**3.9 Алгоритм лексического анализа** 22](#_Toc27686450)

[**3.10 Контрольный пример** 22](#_Toc27686451)

[4. Разработка синтаксического анализатора 23](#_Toc27686452)

[**4.1 Структура синтаксического анализатора** 23](#_Toc27686453)

[**4.2 Контекстно-свободная грамматика, описывающая синтаксис языка** 23](#_Toc27686454)

[**4.3 Построение конечного магазинного автомата** 24](#_Toc27686455)

[**4.4 Основные структуры данных** 25](#_Toc27686456)

[**4.5 Описание алгоритма синтаксического разбора** 25](#_Toc27686457)

[**4.6 Структура и перечень сообщений синтаксического анализатора** 26](#_Toc27686458)

[**4.7 Параметры синтаксического анализатора и режимы его работы** 26](#_Toc27686459)

[**4.8 Принцип обработки ошибок** 26](#_Toc27686460)

[**4.9 Контрольный пример** 27](#_Toc27686461)

[5. Разработка семантического анализатора 28](#_Toc27686462)

[**5.1 Структура семантического анализатора** 28](#_Toc27686463)

[**5.2 Функции семантического анализатора** 28](#_Toc27686464)

[**5.3 Структура и перечень сообщений семантического анализатора** 28](#_Toc27686465)

[**5.4 Принцип обработки ошибок** 29](#_Toc27686466)

[6. Преобразование выражений 31](#_Toc27686467)

[**6.1 Выражения, допускаемые языком** 31](#_Toc27686468)

[**6.2 Польская запись и принцип ее построения** 32](#_Toc27686469)

[**6.4 Контрольный пример** 32](#_Toc27686470)

[7 . Генерация кода 33](#_Toc27686471)

[**7.1 Структура генератора кода** 33](#_Toc27686472)

[**7.3 Статическая библиотека** 34](#_Toc27686473)

[8. Тестирование транслятора 35](#_Toc27686474)

[Приложение А 38](#_Toc27686475)

[Приложение Б 39](#_Toc27686476)

[Приложение В 42](#_Toc27686477)

[Приложение Д 46](#_Toc27686478)

[Приложение Е 47](#_Toc27686479)

[Заключение 51](#_Toc27686481)

**1. Спецификация языка программирования**

## **Характеристика языка программирования**

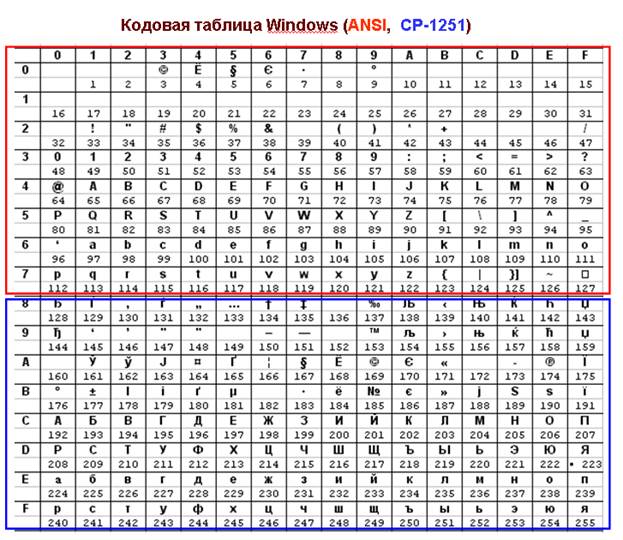
Язык программирования SDE-2019 предназначен для выполнения простых арифметических операция и работы со строками.

Язык программирования SDE-2019 является процедурным, строго типизированным, не объектно-ориентированным, компилируемым.

* 1. **Алфавит языка**

Алфавит языка SDE-2019 основан на кодировке ASCII. Таблица кодировки ASCII представлена на рисунке 1.1.

Исходный код может содержать символы латинского алфавита, цифры десятичной системы счисления, символы пробела, табуляции, перевода строки,символы операторов: “+ - / \* > <” и символы сепараторов: , ; { } ( ) $ ?

Рисунок 1.1 – Таблица кодировки ASCII

* 1. **Применяемые сепараторы**

Применяемые сепараторы в языке SDE-2019, приведены в таблице 1.1

Таблица 1.1 — Применяемые сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение сепаратора |
| ; | Разделитель инструкций |
| { } | Программный блок |
| ( ) | Параметры |
| ’ ’,’\t’ | Служит для разделения. Допускается везде, кроме идентификаторов и ключевых слов |
| , | Разделитель параметров в функции |

* 1. **Применяемые кодировки**

Для написания исходного кода на языке программирования SDE-2019 используется кодировка ASCII, которая представлена на рисунке 1.1.

## **Типы данных**

В языке SDE-2019 есть 2 типа данных: целочисленный и строковый. Описание типов данных, предусмотренных в данным языке представлено в таблице 1.2.

Таблица 1.2 – Типы данных языка SDE-2019

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| little | Фундаментальный тип данных. Предусмотрен для объявления беззнаковых целочисленных данных (2 байта).  Операции:  + бинарная операция сложения;  - бинарная операция вычитания;  \* бинарная операция умножения;  / бинарная операция деления; |
| text | Является строковым типом данных. Предназначен для работы с символами, каждый символ занимает 1 байт. Максимальное количество символов – 255.  Инициализация по умолчанию: все байты имеют значение нулевого символа.  Операции:  + бинарная операция конкатенация; |

## **Преобразование типов данных**

Язык SDE-2019 строго типизированный, преобразование типов данных не поддерживается.

## **Идентификаторы**

Общее количество идентификаторов ограничено максимальным размером таблицы идентификаторов. Идентификаторы должны начинаться только с символов латинского алфавита, могут содержать цифры. Максимальная длина идентификатора равна 10 символам.Если длина идентификатора больше 10, то она автоматически урезается до 10. Идентификаторы, объявленные внутри функционального блока имеют массив из 5 чисел отображающий их область видимости.Данные правила действуют для всех типов идентификаторов.

<буква> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<идентификатор> ::= <буква>{ (<цифра> |<буква> ) }

## **Литералы**

В языке существует 2 типа литералов: целого и строкового типов. Краткое описание литералов представлено в таблице 1.3.

Таблица 1.3 – Описание литералов

|  |  |
| --- | --- |
| Тип литерала | Описание |
| Литералы целого типа | Целочисленные беззнаковые литералы, десятичное и двоичное представления. Литералы только rvalue. |
| Строковые литералы | Состоит из символов латинского алфавита, заключенных в ‘ ‘ (одинарные кавычки). Только rvalue. |

<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<двоичное число> ::= !{0|1}

<целочисленный литерал> ::= (!|цифра){цифра}

## **Объявление данных и область видимости**

В языке SDE-2019 объявление данных начинается с ключевого слова new, указывается тип данных и имя идентификатора. Требуется обязательное объявление переменной перед её использованием.

Примеры: new little a, new text b;

Все переменные в языке SDE-2019 имеют область видимости в виде массива из 5 цифр.

Любая цифра отображает номер по счету от начала программы символа ‘{‘.Если блок вложенный то запись ведется в следующeю цифру массива.

Есть поддержка глобальных переменных.

Примеры области видимости:

new little x;

Глобальная переменная 0 0 0 0 0

{new little x;}

Переменаая в первом блоке 0 1 0 0 0

{{{new little x;}}}

Переменная в нескольких блоках 0 4 5 6 0

Максимальная вложенность блоков равно 4

## **Инициализация данных**

В момент объявления переменных в языке SDE-2019 не происходит инициализация данных. Инициализация значениями в момент объявления не допускается. Виды инициализации представлены в таблице 1.4.

Таблица 1.4 — Способы инициализации переменных

|  |  |
| --- | --- |
| Вид инициализации | Примечание |
| <идентификатор> = <литерал>; | Присваивание переменной значения. |

## **Инструкции языка**

Инструкции языка SDE-2019 представлена в таблице 1.5.

Таблица 1.5 – Инструкции языка

|  |  |
| --- | --- |
| Инструкция | Форма записи |
| Объявление переменной | new <тип данных> <идентификатор> |
| Объявление функции | <тип данных> function <идентификатор> ({<тип данных> <идентификатор>,})  {  < программный блок>  return <идентификатор>|<литерал>.  } |
| Вызов функции | <идентификатор> (<идентификатор>|<литерал>, …) |
| Присвоение значения | <идентификатор> = <идентификатор>|<литерал>; |
| Печать данных | output <литерал>|<идентификатор> |
| Функции стандартной библиотеки | textlenght (text) — размер строки  copytxt(text, text) — копирует вторую строку в первую |
| Возвращаемое значение | return <литерал>|<идентификатор> |

## **Операции языка**

Операция сложения применима к строковым и целочисленным типам данных. При строковых типах данных происходит конкатенация строк. Остальные операции в языке программирования применимы исключительно к целочисленным типам данных.

Наибольшую приоритетность арифметических операций имеют операции сложения и деления, а сложение и вычитание меньшую. При одинаковом приоритете первой выполнится операция расположенная левее. Изменить приоритетность можно с помощью круглых скобок.

Операции в языке программирования SDE-2019 применимые к целочисленным и строковым типам данных приведены в таблице 1.6.

Таблица 1.6 — Операции языка

|  |  |
| --- | --- |
| Тип | Операторы |
| Арифметические | () – приоритетность операций  + ̶ сложение  -  ̶ вычитание  \* ̶ умножение  / ̶ деление  %-остаток от деления |
| Строковые | + – конкатенация |
| Логические | = – равенство  > – больше  < – меньше |

## **Выражения и их вычисление**

Предусмотрены следующие правила составления выражений:

* выражения читаются слева направо и записываются в одну строку;
* реализация выражений происходит с помощью обратной польской записи;
* для изменения приоритета операция используются круглые скобки.

## **Программные конструкции языка**

Язык SDE-2019 имеет следующие программные конструкции:

* Функции
* Условные конструкции
* Конструкция цикла

Основные программные конструкции языка SDE-2019 представлены в таблице 1.7.

Таблица 1.7 — Основные конструкции языка

|  |  |
| --- | --- |
| Конструкция | Реализация |
| Главная функция  (точка входа) | start  {  <программный блок>  }; |
| Функции | <тип данных> function <идентификатор>  (<тип данных> <идентификатор>, …)  {  <программный блок>  return <идентификатор>|<литерал>.  }; |
| Условный оператор | check(<литерал>|<идентификатор>  ( = | < | > )<литерал>|<идентификатор>)?  {  <программный блок>  }  not  {  <программный блок>  } |
| Оператор цикла | from (<литерал>|<идентификатор>)to (<литерал>|<идентификатор>)$  {  } |

## **Область видимости идентификаторов**

Область видимости нумеруется сверху вниз. В SDE-2019 требуется обязательное объявление переменной перед её использованием.Объявление переменной может быть как в блоке, так и вне блока. Имеется возможность объявления одинаковых переменных в разных блоках.Область видимости для каждой переменной является массив из 5 цифр

## **Семантические проверки**

В языке программирования SDE-2019 выполняются следующие семантические проверки:

1. Наличие функции start – точкa входа в программу;

2. Уникальность функции start;

3. Переопределение идентификаторов;

4. Использование идентификаторов без их объявления;

5. Проверка соответствия типа функции и возвращаемого параметра;

6. Правильность передаваемых в функцию параметров: количество, типы;

7. Правильность строковых выражений;

8. Превышение размера строковых литералов;

9. Правильность составленного цикла.

10. Правильность составления выражений

11. Правильность переданных параметров в цикл

12. Правильность составленного условия выражения.

13. Правильность переданных параметров в условное выражение

14. Проверка на максимальное количество операдов в выражении

15. Проверка на вложенный цикл в цикл или условное выражение в условное выражение

16. Проверка на деление числа на 0

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

Транслированный код использует две области памяти. В сегмент констант заносятся все литералы. В сегмент данных заносятся переменные и копии строковых литералов. Локальная область видимости в исходном коде определяется постфиксом который имеет вид 5 подряд идущих цифр в десятичной системе.

## **Стандартная библиотека и её состав**

В языке SDE-2019 предусмотрена стандартная библиотека. Функции, входящие в состав библиотеки, описаны в табл. 1.7. Стандартная библиотека подключается автоматически на этапе генерации кода.

Таблица 1.7 - Функции стандартной библиотеки языка SDE-2019

|  |  |
| --- | --- |
| Функция | Описание |
| copytxt (text a, text b) | Данная функция строкового типо принимает два параметра (две строки).  Копирует значениестроки b в строку a.В случае если а – литерал , используется его копия из сегмента данных; |

Продолжение таблицы 1.7

|  |  |
| --- | --- |
| textlenght(text a) | Данная функция целочисленного типа принимает один параметр(строку). Функция вычисляет размер строки и возвращает результат. |

## **Ввод и вывод данных**

Ввод данных языком программирования, SDE-2019 не поддерживается.

Для вывода данных используется функция output(<имя идентификатора>|<литерал>). Пример: output(a);

## **Точка** **входа**

Функция точки входа в языке программирования SDE-2019 представлена в таблице 1.10.

Таблица 1.10 — Точка входа

|  |  |
| --- | --- |
| Конструкция | Реализация |
| Главная функция  (точка входа) | start  {  <программный блок>  }; |

## **Препроцессор**

Препроцессор в языке SDE-2019 отсутствует.

## **Соглашения о вызовах**

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

– все параметры функции передаются через стек;

– память высвобождает вызываемый код;

– занесение в стек параметров идёт справа налево

## **Объектный код**

Код языка SDE-2019 транслируется в язык ассемблера.

## **Классификация сообщений транслятора**

Генерируемые транслятором сообщения определяют степень его информативности, то есть сообщения транслятора должны давать максимально полную информацию о допущенной пользователем ошибке при написании программы. Сообщения транслятора приведены в таблице 1.10, а также в приложении А.

Таблица 1.10 Классификация ошибок(диапазон)

|  |  |
| --- | --- |
| Префикс ошибки | Описание ошибки |
| ### SYSTEM: | ### - код ошибки. Сообщение, генерируемое при критической ошибке системы. Диапазон: 0-99 |
| ### PARAMETERS: | ### - код ошибки. Сообщение, генерируемое при ошибке ввода параметров. Диапазон: 100-109 |
| ### LEXICAL: | ### - код ошибки. Сообщение, генерируемое при ошибке на этапе лексического анализа. Диапазон: 110-129 |
| ### SYNTAX: | ### - код ошибки. Сообщение, генерируемое при ошибке на этапе синтаксического анализа. Диапазон: 130-149 |
| ### SEMANTICS: | ### - код ошибки. Сообщение, генерируемое на этапе семантического анализа. Диапазон: 150-179 |
| ### ADDITIONAL: | ### - код ошибки. Сообщение, генерируемое на других этапах.Диапазон: 500-599 |

## **Контрольный пример**

new text out;

new little x;

new little sum;

little function addn()

{

x = textlenght(out) -1 ;

return x;

};

little function show(little u)

{

out = ' ';

x = u % 2;

from(0)to(u)$

{

check(u = 0)?

{

out = out + '@';

}

not

{

out = out + '\*';

}

}

x = addn();

sum = sum + x;

output(out);

output(x);

return 0;

};

start

{

from(1)to(9)$

{

x = show(buffer);

}

from(9)to(1)$

{

x = show(buffer);

}

output(' ');

output('Sum');

output(sum);

return 0;

};

# **2. Структура транслятора**

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

Исходный код, написанный на языке программирования SDE-2019, является для транслятора входными данными.

Как выходные данные используется объектный код и протоколы работы транслятора, описанные в пункте 2.3.

Принцип их взаимодействия представлен на рисунке 2.1.

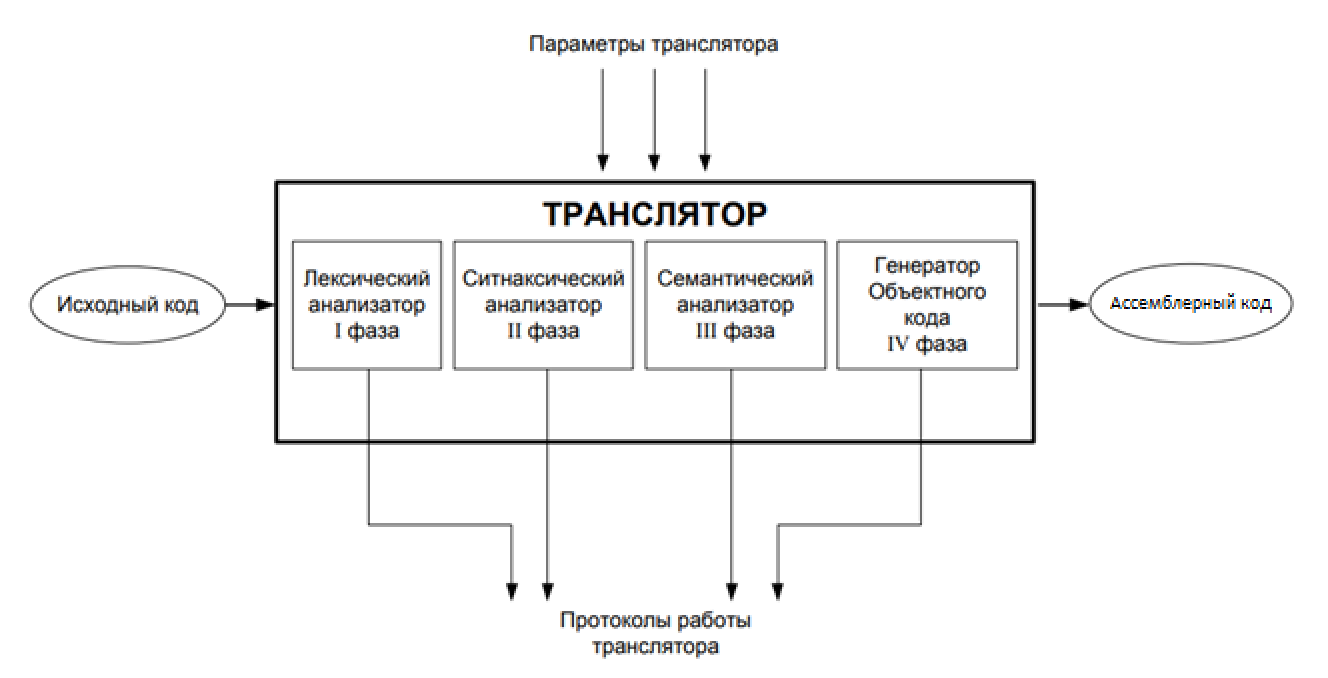


Рисунок 2.1 – Структура транслятора

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

Синтаксический анализатор – часть компилятора, выполняющая синтаксический анализ, то есть проверку исходного кода на соответствие правилам грамматики. Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов. Выходной информацией является дерево разбора.

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

Генератор кода – этап транслятора, выполняющий генерацию ассемблерного кода на основе полученных данных на предыдущих этапах трансляции. Генератор кода принимает на вход таблицы идентификаторов и лексем и транслирует код на языке SDE-2019, прошедший все предыдущие этапы, в код на языке Ассемблера. Более полно описан в главе 7.

## **2.2 Перечень входных параметров транслятора**

Входные параметры представлены в таблице 2.1.

Таблица 2.1 Входные параметры транслятора языка SDE-2019

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание параметра | Значение по умолчанию |
| -in:”имя исходного файла” | Файл с исходным кодом на языке SDE-2019, имеющий расширение .txt | Нет |
| -log:”имя файла лога” | Файл журнала для вывода протоколов работы программы. | Значение по умолчанию: “имя исходного файла”.log |
| -out: ”имя выходного файла” | Выходной файл – результат работы транслятора. Содержит исходный код на языке асемблера. | Значение по умолчанию:  “имя исходного файла”.asm |

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

Таблица с перечнем протоколов, формируемых транслятором языка

SDE-2019 и их назначением представлена в таблице 2.2

Таблица 2.2 Протоколы, формируемые транслятором языка SDE-2019

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |
| Файл журнала, заданный параметром "-log:" | Файл с протоколом работы транслятора языка программирования SDE-2019. Содержит таблицу лексем, таблицу идентификаторов, маленькую таблицу лексем,дерево разбора a также информацию об ошибках и исходных данных |

Продолжение таблицы 2.2

|  |  |
| --- | --- |
| Выходной файл, заданный параметром "-out:" | Результат работы программы – файл, содержащий исходный код на языке ассемблера. |
| Выходной файл  trace.txt | Файл содержит подробную информацию о трассировке при выполнении синтаксического анализа |

**3. Разработка лексического анализатора**

## **3.1 Структура лексического анализатора**

Лексический анализатор – часть транслятора, выполняющая лексический анализ. Лексический анализатор принимает обработанный и разбитый на отдельные компоненты исходный код на языке SDE-2019. На выходе формируется таблица лексем и таблица идентификаторов. Структура лексического анализатора представлена на рисунке 3.1



Рисунок 3.1 Структура лексического анализатора SDE-2019

## **3.2 Контроль входных символов**

Таблица для контроля входных символов представлена на рисунке 3.2

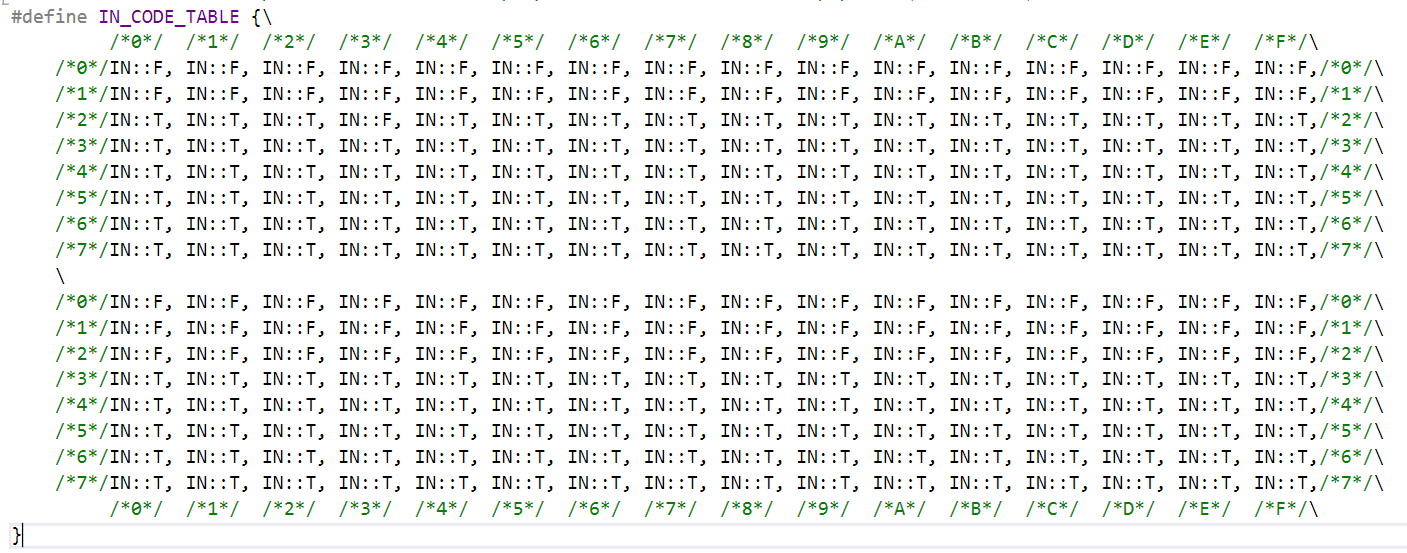


Рисунок 3.2. Таблица контроля входных символов

Принцип работы таблицы заключается в соответствии значения каждому элементу в шестнадцатеричной системе счисления значению в таблице ASCII.

Описание значения символов: T – разрешённый символ, F – запрещённый символ

## **3.3 Удаление избыточных символов**

Избыточными символами являются символы табуляции,пробелы и символы перехода на новую строку. До лексического анализа исходный текст разбивается на лексемы игнорируя избыточные символы

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

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

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

|  |  |  |
| --- | --- | --- |
| Тип | Слова | Лексема |
| Ключевые слова | new | n |
| little | t |
| text | t |
| function | f |
| output | o |
| from | m |
| to | e |
| return | r |
| start | s |
| ckeck | c |
| not | j |
| textlenght | z |
| copytxt | z |
| Данные | Идентификатор | i |
| Целочисленный литерал | l |
| Строковый литерал | l |
| Сепораторы | ; | ; |
| , | , |
| { | { |
| } | } |
| ( | ( |
| ) | ) |
| ? | ? |
| $ | $ |

Продолжение таблицы 3.1

|  |  |  |
| --- | --- | --- |
| Операторы | + | + |
| - | - |
| \* | \* |
| / | / |
| = | = |
| % | % |
| > | > |
| < | < |

Пример реализации таблицы лексем и таблицы идентификаторов представлен

в приложении Б.

В приложении  В находятся конечные автоматы, соответствующие лексемам языка SDE-2019.

## **3.5 Основные структуры данных**

Основные структуры таблиц лексем и идентификаторов данных языка SDE-2019, используемых для хранения, представлены в приложении В. В таблице лексем содержится лексема, её номер, полученный при разборе, номер строки в исходном коде а также дополнительная информация в поле value. В таблице идентификаторов содержится имя идентификатора, номер в таблице лексем, тип данных, смысловой тип идентификатора и его значение.

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

Перечень сообщений лексического анализатора представлен в таблицу 3.2

Таблица 3.2 – Перечень ошибок лексического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 110 | LEXICAL: Превышен максимльный размер таблицы лексем |
| 111 | LEXICAL: Таблица лексем переполнена |
| 112 | LEXICAL: Не найден конец строкового литерала или строковый литерал является пустым |
| 113 | LEXICAL: Неправильно объявление числового литерал |
| 114 | LEXICAL: Превышен максимальный размер таблицы идентификаторов |
| 115 | LEXICAL: Таблица идентификаторов переполнена |
| 116 | LEXICAL: Ошибка лексического анализа |
| 117 | LEXICAL: Имя идентификатора задано неверно |
| 118 | LEXICAL: Превышен максимальный размер строкового литерала |
| 119 | LEXICAL: Превышено максимальное количество вложенных блоков |

## **3.7 Принцип обработки ошибок**

В случае обнаружения ошибки, которая не позволяет работать анализаторам или генератору правильно функционировать, транслятор прекращает свою работу и в log-файл записывается ошибка.

## **3.8 Параметры лексического анализатора**

Входным параметром лексического анализа является исходный текст. В начале лексического анализа текст очищается от избыточных символов и полученная после чтения входного файла на этапе проверки исходного кода на допустимость символов.

## **3.9 Алгоритм лексического анализа**

* проверяет входной поток символов программы на исходном языке на допустимость
* разбивает на лексемы(токены) игнорируя избыточные символы.
* Для каждой лексемы пытается определить ее тип и параметры.
* Если определить тип из параметры удалось то заносит информацию о лексеме в таблицу лексем и таблицу идентификаторов, и алгоритм возвращается к первому этапу;
* формирует файлы таблиц и заносит их в протокол;
* при неуспешном распознавании выдается сообщение об ошибке.

Распознавание цепочек основывается на работе конечных автоматов.Если лексический анализатор определяет ,что это может быть ключевое слово,то он запускает функцию Update, которая должна вернуть специальное значение.Функия Update посылает лексему определенному автомату.Автомат возращает значение -1 ,если цепочка не разобрана, или специальное значение, если цепочка была разобрана. Если специальное значение сходится со значением предпологаемым лексическим анализатором ,то это означает, что данная лексема это ключевое слово

## **3.10 Контрольный пример**

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

# **4. Разработка синтаксического анализатора**

## **4.1 Структура синтаксического анализатора**

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



Рисунок 4.1 – Структура синтаксического анализатора

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

В синтаксическом анализаторе транслятора языка SDE-2019 используется контекстно-свободная грамматика , где

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

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

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

S – начальный символ грамматики, являющийся нетерминалом.

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

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

Грамматика языка SDE-2019 представлена в приложении Б.

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

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

Таблица 4.1 – Перечень и описание нетерминальных символов SDE-2019

|  |  |
| --- | --- |
| Нетереминалы | Описание |
|  | Правила, описывающие общую структуру программы |
|  | Порождает правила, описывающие инструкции языка |
|  | Порождает правила, описывающие выражения |
|  | Порождает правила, описывающие арифметические действия |
|  | Порождает правила, описывающие формальные параметры функции |
|  | Порождает правила, описывающие фактические параметры функции |
|  | Порождает правила, описывающие  параметры передаваемые в условный оператор и оператор цикла |
|  | Порождает правила, описывающие  логические операторы в условной конструкции |

## **4.3 Построение конечного магазинного автомата**

Принцип действия конечного магазинного автомата представлен на рисунке 4.2.



Рис. 4.2 - МП-автомат

Формальное описание МП-автомата:



 - множество состояний;

 - алфавит входных символов;

 - специальный алфавит магазинных символов;

-функция переходов автомата , где  - множество подмножеств ;

 - начальное состояние автомата;

- начальное состояние магазина (маркер дна);

- множество конечных состояний.

Конфигурация (текущее состояние автомата) описывается тройкой , где - текущее состояние автомата, - остаток цепочки,  - цепочка-содержимое магазина.

Начальное состояние , - начальное состояние автомата,  - входная цепочка, - маркер дна магазина.

Цепочка  является допустимой (распознается) автоматом , если  и .

Работа автомата 

1. состояние автомата 
2. читает символ  находящийся под головкой (сдвигает ленту);
3. не читает ничего (читает , не сдвигает ленту);
4. из  определяет новое состояние , если  или .
5. читает верхний (в стеке) символ  и записывает цепочку  т.к. , при этом, если , то верхний символ магазина просто удаляется.

работа автомата заканчивается 

## **4.4 Основные структуры данных**

Основные структуры данных синтаксического анализатора включают в себя структуру магазинного автомата и структуру грамматики Грейбах, описывающей правила языка SDE-2019. Данные структуры представлены в приложении Г.

## **4.5 Описание алгоритма синтаксического разбора**

Входные символы и лексемы в форме Грейбах находятся в ленте на входе конечного автомата.

1) Если лента не пустая, переходим далее следующему пункту, иначе переходим к пункту 5.

2) Если на верхушке магазина нетерминальный символ.

2.1) Если есть такое правило, то переходим к следующему пункту.

2.1.1) Если цепочка есть, возвращаем NS\_OK. Переходим к пункту 4.

2.1.2) Иначе восстанавливаем состояние. Переходим к пункту 4.

2.2) Иначе возвращаем ошибку. Переход к пункту 4.

3) Если на верхушке терминал и он совпадает с символом на ленте, то удаляем его из стека и продвигаем ленту. Переход к пункту 4.

4) Повторяем шаг, переходим к пункту 1.

5) Конец работы.

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

Перечень сообщений синтаксического анализатора представлен в таблице 4.2.

Таблица 4.2 – Перечень ошибок синтаксического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 130 | SYNTAX: Неверная структура программы |
| 131 | SYNTAX: Ошибка в структуре блока |
| 132 | SYNTAX: Ошибочный оператор |
| 133 | SYNTAX: Ошибка в параметрах функции |
| 134 | SYNTAX: Ошибка в параметрах вызываемой функции |
| 135 | SYNTAX: Ошибочный оператор |
| 136 | SYNTAX: Ошибка в синтаксическом анализе. За подробной информацией обратитесь к файлу trace |
| 137 | SYNTAX: Ошибка в параметрах конструкции |
| 138 | SYNTAX: Ожидался условный оператор |

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

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

Выходными параметрами являются трассировка прохода таблицы лексем,вывод в файл trace, и правила разбора, которые выводятся в log файл.

## **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 в процессе анализа находятся более трёх ошибок, то анализ останавливается. При успешном прохождении синтаксического анализатора в протокол будет выведено дерево разбора.

Подробная трассировка прохода с описанием ошибок будет находится в файле trace.

## **4.9 Контрольный пример**

Пример разбора синтаксическим анализатором исходного кода на языке SDE-2019 представлен в приложении Д. Дерево разбора исходного кода также представлено в приложении Д.

# **5. Разработка семантического анализатора**

## **5.1 Структура семантического анализатора**

В SDE-2019 cемантический анализ разбит на 3 части.Первая происходит во время лексического анализа.Вторая часть запускается перед преоразованием в польскую запись.Третья часть запускается непосредственно после изменения таблицы лексем и таблицы идентификаторов польской записью. Анализа реализуется в виде отдельных проверок текущих ситуаций в конкретных случаях.

## **5.2 Функции семантического анализатора**

Семантический анализатор имеет две функции.Первая(до выполнения польской записи) проверяет основную семантику связанную со структурой программы:

* Проверка на существование точки входа
* Проверка на единственность точки входа
* Проверяет колличесво и типы параметров передаваемых в функцию
* Проверяет на максимальное колличество параметров
* Проверяет на максимальное колличество операндов в выражении

Вторая функия (после выполнения польской записи) проверяет семантику выражений:

* Проверяет на соответсвие типов rvalue и lvalue
* Проверяет правильность составления операторов цикла и условных операторов

Реализованно через стек.

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

Сообщения, формируемые семантическим анализатором, представлены на рисунке 5.1.

таблица 5.1 – Перечень сообщений семантического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 150 | SEMANTICS: Функция start должна быть определена только 1 раз |
| 151 | SEMANTICS: Определение функции start не обнаруженo |
| 152 | SEMANTICS: Повторное объявление переменной в данной области видимости |
| 153 | SEMANTICS: Используется не объявленная переменная |
| 154 | SEMANTICS: Неверное объявление переменной |
| 155 | SEMANTICS: Неверные типы параметров |
| 156 | SEMANTICS: Неверное количество параметров |
| 157 | SEMANTICS: Несоответсвие типов |

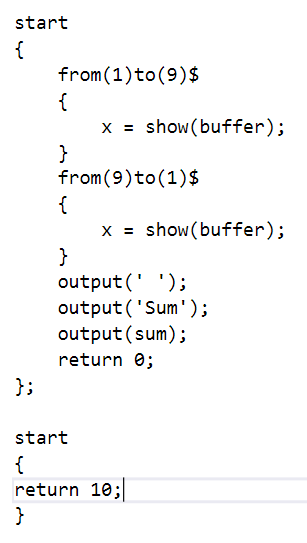
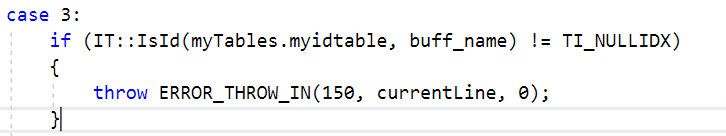
Продолжение таблицы 5.1

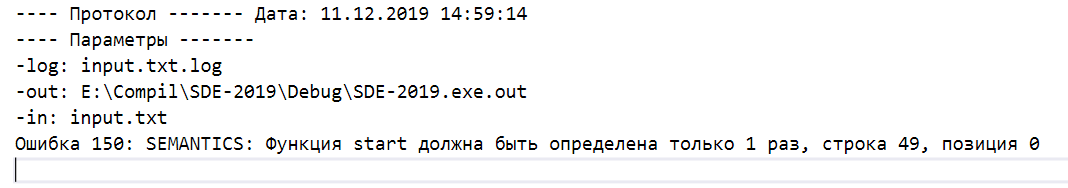
|  |  |  |
| --- | --- | --- |
| 158 | | SEMANTICS: Превышено максимальное колличество передаваемых параметров при определении функции |
| 159 | | SEMANTICS: Превышено максимальное колличество передаваемых параметров при вызове функции |
| 160 | | SEMANTICS: Функция не возращает значение |
| 161 | SEMANTICS: Тип возращаемого значения не соответсвует типу функции | |
| 162 | SEMANTICS: Превышено максимальное количество опердов для выражения | |
| 163 | SEMANTICS: Параметры цикла могут быть только типа little | |
| 164 | SEMANTICS: К операндам типа text применим только оператор + | |
| 165 | SEMANTICS: Вложенные циклы не поддерживаются | |
| 166 | SEMANTICS: Параметры условного оператора должны быть типа little | |

## **5.4 Принцип обработки ошибок**

В случае обнаружения ошибки, которая не позволяет правильно сгенерировать код в ассемблер, транслятор прекращает свою работу и в log-файл записывается ошибка.

**5.5 Контрольный пример**



# **6. Преобразование выражений**

**6.1 Выражения, допускаемые языком**

В языке SDE-2019 допускаются выражения с использованием числовых идентификаторов и литералов. Также предусмотрены следующие арифметические операции:

* сложения: «+» ;
* вычитания: «-» ;
* умножения: «\*» ;
* деления: «/» ;
* остаток от деления: «%» ;

И логические операции:

* равенство: «=» ;
* больше: «>» ;
* меньше: «<» ;

Также есть возможность изменять приоритет выполнения арифметических операций при помощи скобок:

* «(»;
* «)»;

За счет заключения операции в скобки происходит повышение его приоритета, а значит при вычислении всего выражения операция с более высоким приоритетом будет вычисляться раньше операции с меньшим приоритетом.

Возможна конкатенация строк посредством оператора сложения. Приоритетность операций представлена в таблице 6.1. Чем выше число, тем выше и приоритет.

Таблица 6.1 - Приоритетность операций

|  |  |
| --- | --- |
| Операция | Приоритет |
| «(» | +2 |
| «)» | -2 |
| «+», «-» | 0 |
| «\*», «/»,«%» | 1 |

**6.2 Польская запись и принцип ее построения**

Польская запись -форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. Приоритетность операций приведена в таблице 6.1. В SDE-2019 используется следующий принцип построения польской записи:

1. Переписываются все иденитфикаторы и литералы в порядке в котором они идут в исходном выражении.
2. Записываются все операторы и индексируются приоритетом.
3. Если остались необработанные операторы, то выбираем необработынный оператор с наивысшим приоритетом. Если у двух опреаторов одинаковый приоритет, то выбирается первый слева. Если равно 0 то к шагу 7.
4. Обозначает элементы слева и справа как используемые.
5. Оператор сдвигается вправо до тех пор пока символ слева не будет с пометкой не обработанный.
6. Переход к шагу 3.
7. Конец алгоритма

**6.3 Программная реализация обработки выражений**

После этапов лексического , синтаксического и первой функции семантического анализа происходит преобразование в польскую запись

**6.4 Контрольный пример**

Контрольный пример разбора выражения содержится в таблице 6.2.

Таблица 6.2 - Разбор выражения

|  |  |  |
| --- | --- | --- |
| Полное выражение | Идентификаторы и литералы  (0-неиспользованный,1-использованый) | Операторы(0-неиспользованный,1-использованый;приоритет ) |
| (4+x)/8 | 4(0) x(0) 8(0) | +(0;2) /(0;1) |
| 4+x/8 | 4(0) x(0) 8(0) | +(0;2) /(0;1) |
| 4x+/8 | 4(1) x(1) 8(0) | +(1;2) /(0;1) |
| 4x+8/ | 4(1) x(1) 8(1) | +(1;2) /(1;1) |

# **7 . Генерация кода**

**7.1 Структура генератора кода**

Заключительным этапом трансляции языка SDE-2019 является генерация кода. Таблицы лексем и идентификаторов подаются Генератор принимает на вход таблицы лексем и идентификаторов, полученные в результате лексического анализа. Выходной файл на языке ассемблера, который будет являться результатом работы транслятора, в соответствие с таблицей лексем. В случае возникновения ошибок генерация кода не будет осуществляться. Структура генератора кода SDE-2019 представлена на рисунке 7.1.



Рисунок 7.1 – Структура генератора кода

**7.2 Представление типов данных в памяти**

Элементы таблицы идентификаторов расположены сегментах .data и .const языка ассемблера. Соответствия между типами данных идентификаторов на языке SDE-2019 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка SDE-2019 и языка ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке SDE-2019 | Тип идентификатора на языке ассемблера | Пояснение |
| Little | SDWORD | Хранит целочисленный тип данных. |
| Text | BYTE 255 | Хранит массив байл заполненных символом \0. Представляет строковый тип. |

**7.3 Статическая библиотека**

В языке SDE-2019 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++. Объявление функций статической библиотеки генерируется автоматически.

Вызовы стандартных функций доступны там же, где и вызов пользовательских функций. Также в стандартной библиотеке реализованы функции для управления выводом а также для работы со строками, недоступные конечному пользователю. Эти функции представлены в таблице 1.9.

Таблица 1.9 - Дополнительные функции стандартной библиотеки

|  |  |
| --- | --- |
| Функция на языке С++ | Описание |
| char\* cleartxt (char\*) | Функции для очистки строки. |
| int outtxt(char\*) | Функции для вывода в стандартный поток значения строкового идентификатора/литерала. |
| int outlit (int) | Функции для вывода в стандартный поток значения для целочисленного идентификатора/литерала. |
| char\* txtcon(char\* str1, char\* str2) | Строковая функция. Возвращает результат конкатенации строк a, b, записанный по адресу a. |
| int sleep | Функция останавливает выполение программы до взаимодействия с пользователем |

**7.4 Оссобенности алгоритма генерации кода**

В языке SDE-2019 генерация кода строится на основе таблиц лексем и идентификаторов. Общая схема работы генератора кода представлена на рисунке 7.2

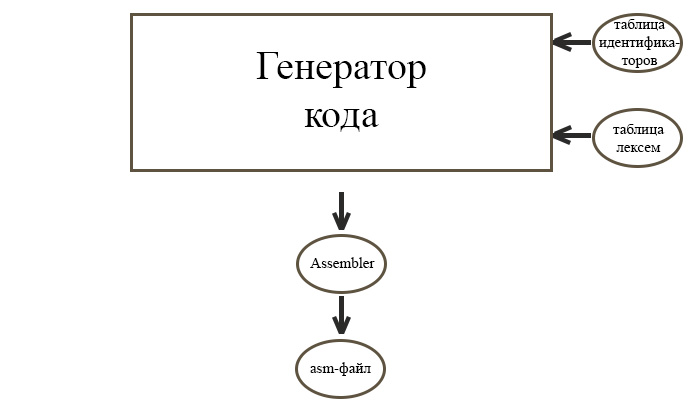


Рисунок 7.2 - Структура генератора кода

**7.5 Контрольный пример**

Результат генерации ассемблерного кода на основе контрольного примера из приложения А приведен в приложении Е.

# **8. Тестирование транслятора**

В данной главе описаны возможные ошибки, возникающие на различных этапах работы транслятора. Результат тестирования представлены в таблицая 8.1,8.2,8.3

Таблица 8.1 - Тестирование транслятора(лексические ошибки)

|  |  |
| --- | --- |
| Исходный код | Генерируемая ошибка |
| start  {  new text a ;  a= ‘string;  } | Ошибка 112: LEXICAL: Не найден конец строкового литерала или строковый литерал является пустым |
| start  {  new little a ;  a= -12fs;  } | Ошибка 113: LEXICAL: Неправильно объявление числового литерала |
| start  {  new little 2a;  } | Ошибка 117: LEXICAL: Имя идентификатора задано неверно |
| start  {  {{{{  new little a;  a = 2;  }}}}  } | Ошибка 119: LEXICAL: Превышено максимальное количество вложенных блоков |

Таблица 8.2 (синтаксические ошибки)

|  |  |
| --- | --- |
| Исходный код | Генерируемая ошибка |
| start  {  …=2; | Ошибка 130: SYNTAX: Неверная структура программы |

Продолжение таблицы 8.2

|  |  |  |
| --- | --- | --- |
| …=2;; | Ошибка 131: SYNTAX:Ошибочный оператор | |
| start  {  …=++2;  } | Ошибка 132: SYNTAX:Ошибка в выражении | |
| little function a(little little a)  {  new little a;  }; | Ошибка 133: SYNTAX: Ошибка в параметрах функции" | |
| little function a()  {  return 2;  };  start  {  new little d;  d= a(new);  } | | Ошибка 134: SYNTAX: Ошибка в параметрах вызываемой функции |

Таблица 8.3(семантические ошибки)

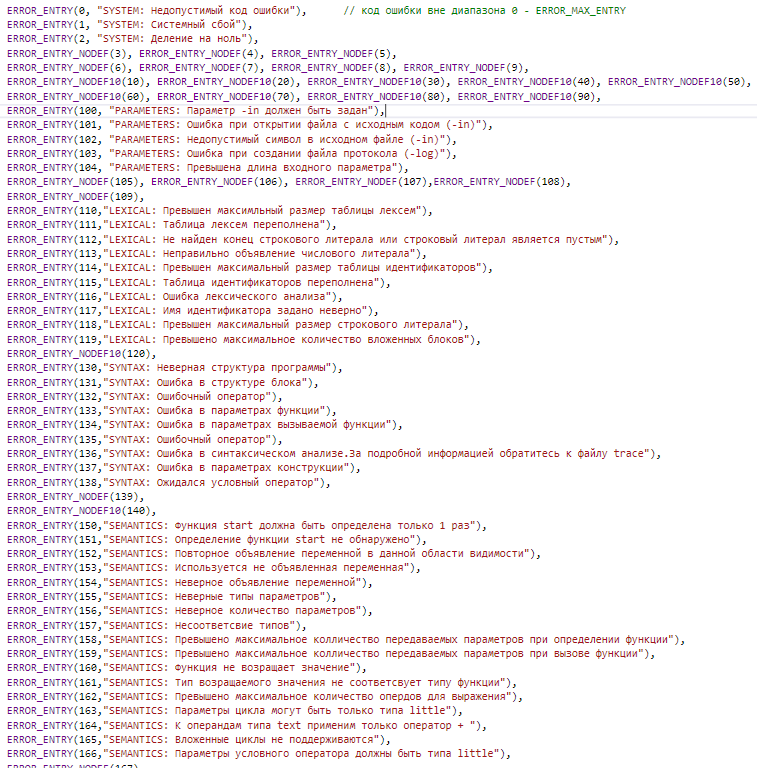
|  |  |
| --- | --- |
| Исходный код | Генерируемая ошибка |
| little function a(little a)  {  return 2;  }; | Ошибка 151: SEMANTICS: Определение функции start не обнаруженo |
| little function a(little a)  {  return 2;  };  start  {  new little и; b= a("string");  } | Ошибка 155: SEMANTICS: Неверные типы параметров |
| little function a()  …  new little b; b= a(2); | Ошибка 156: SEMANTICS: Неверное количество параметров |

Продолжение таблицы 8.2

|  |  |
| --- | --- |
| start  {  new text a = 2;  } | Ошибка 157: SEMANTICS: Несоответсвие типов |
| start  {  text "string" / "string";  } | Ошибка 164: SEMANTICS: К операндам типа text применим только оператор + |
| text function c()  {  return 2;  }; | Ошибка 161: SEMANTICS: Тип возращаемого значения не соответсвует типу функции |
| text function c()  {  new little x;  }; | Ошибка 160: SEMANTICS: Функция не возращает значение |

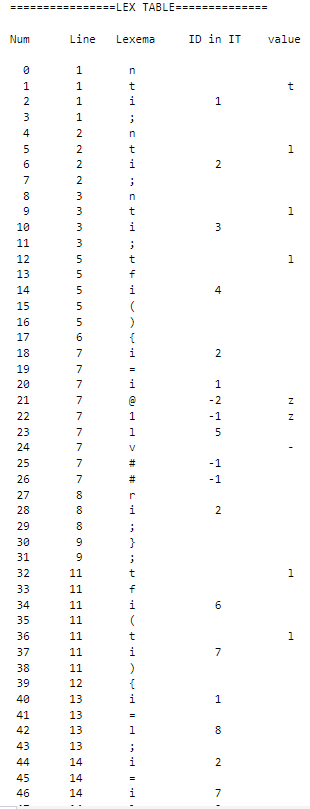
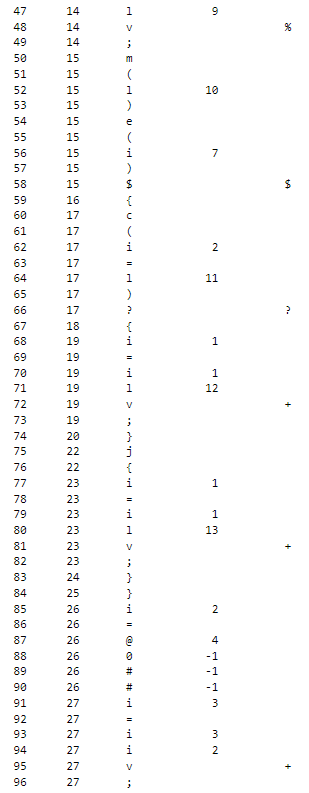
# 

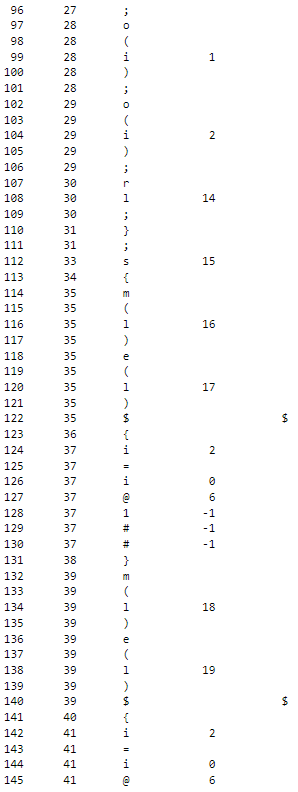
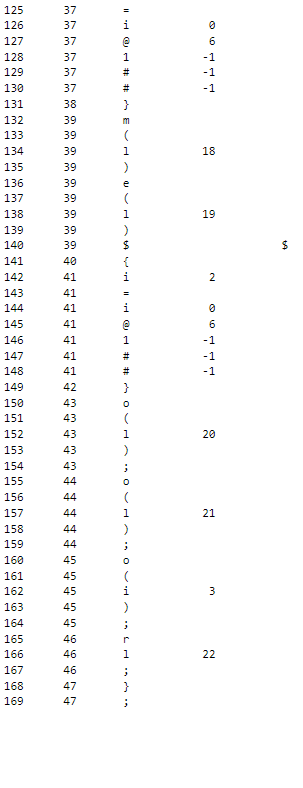
# **Приложение А**

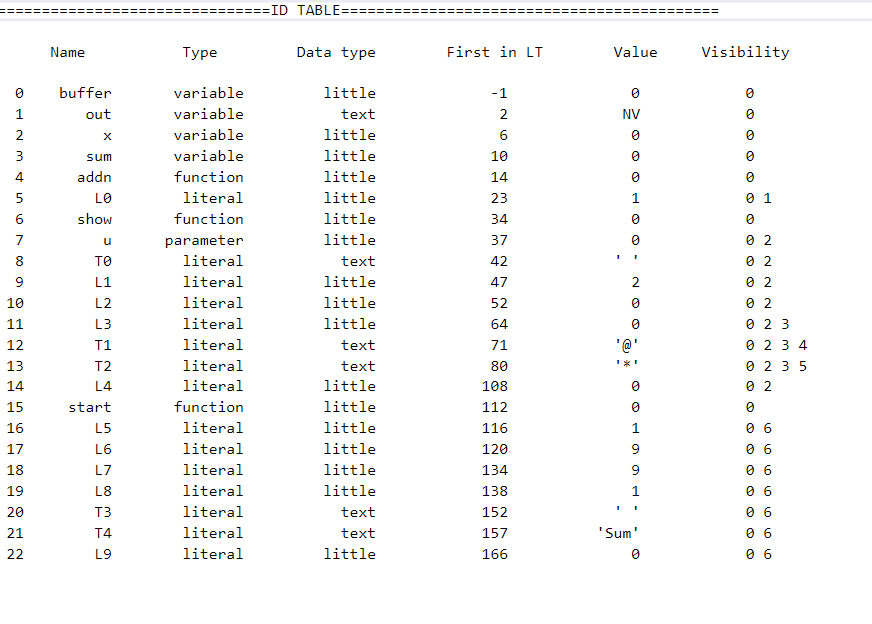


# **Приложение Б**

Таблицы лексем и идентификаторов

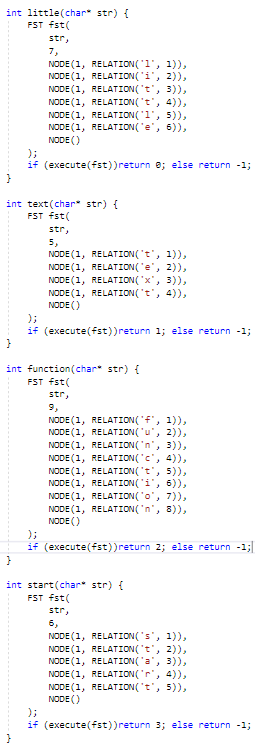
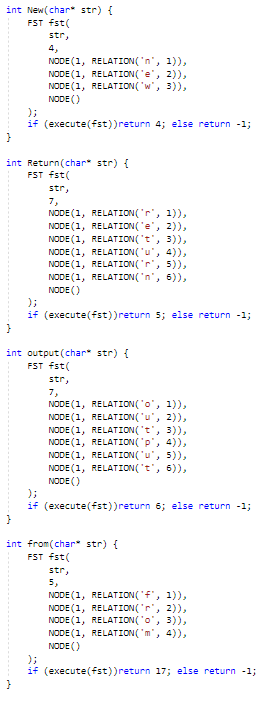
 

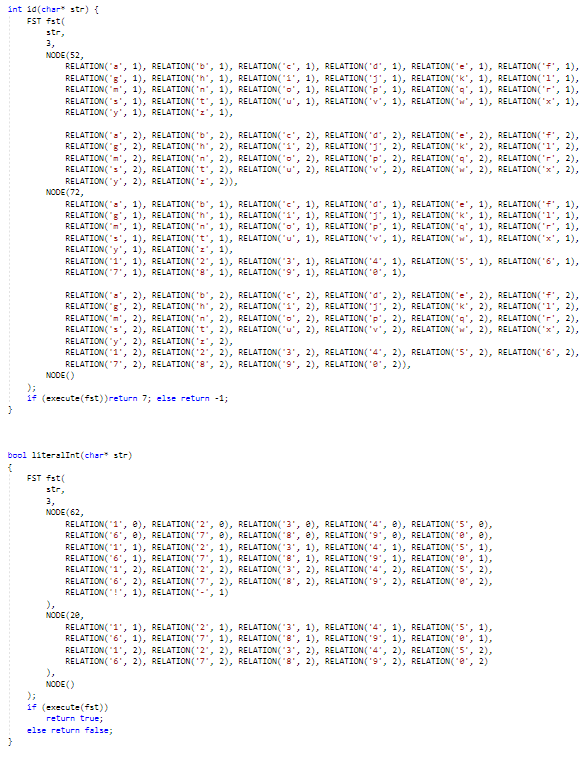
 

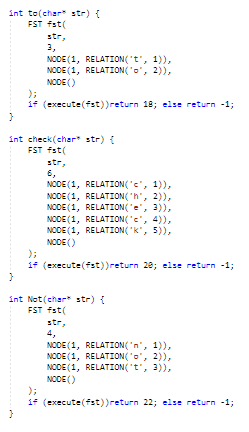
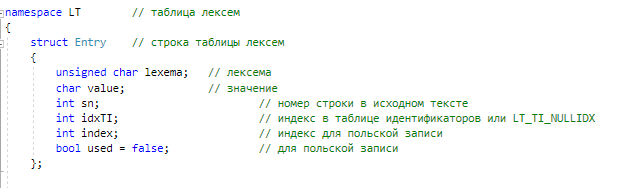


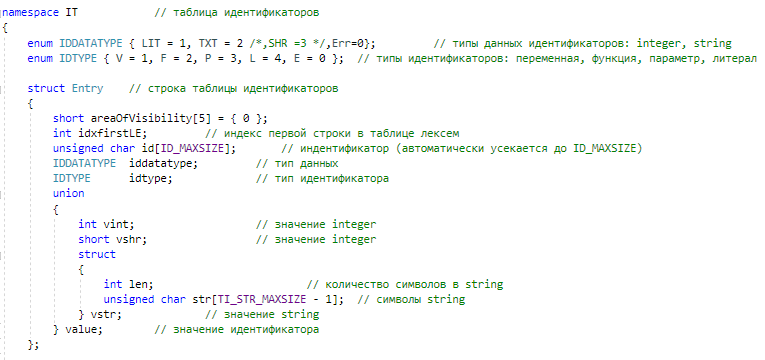
# **Приложение В**

Конечные автоматы и структуры для таблиц

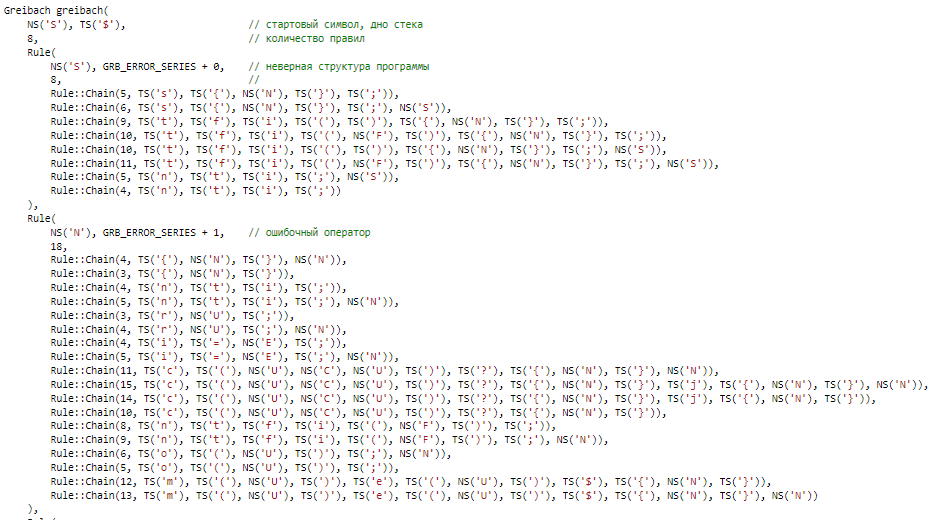


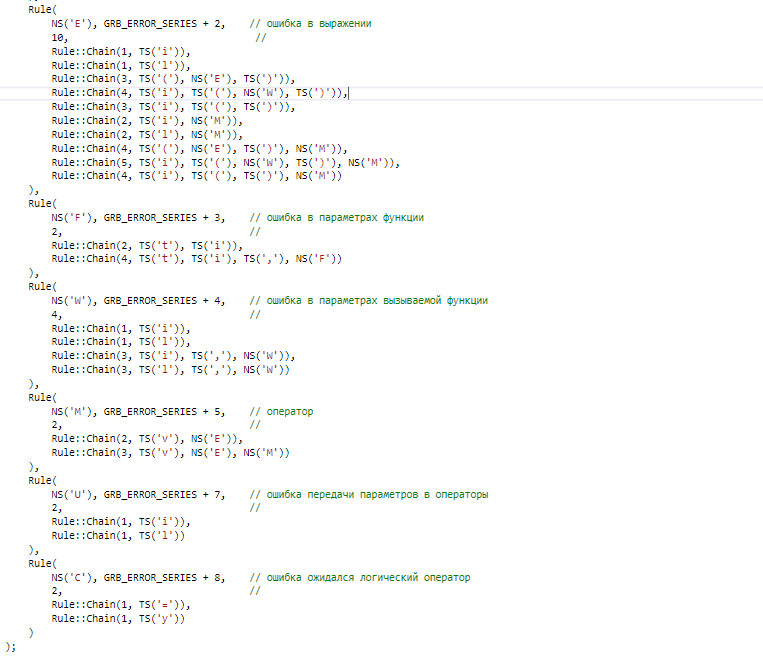
 



**Приложение Г**

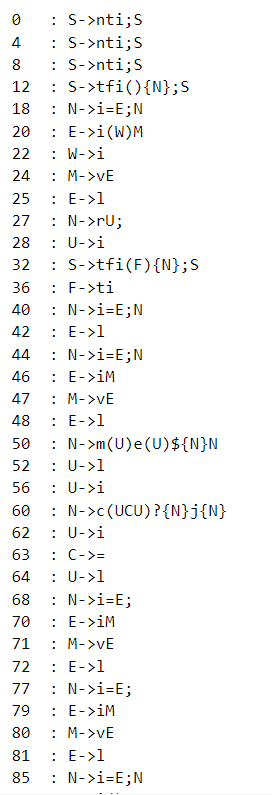
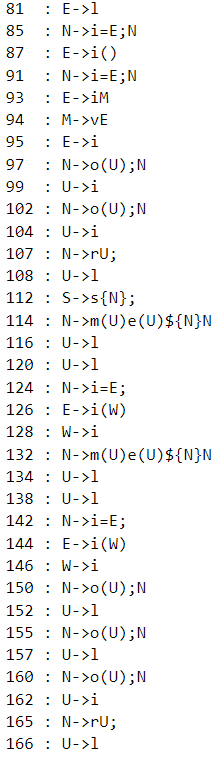
Синтаксические правила SDE-2019





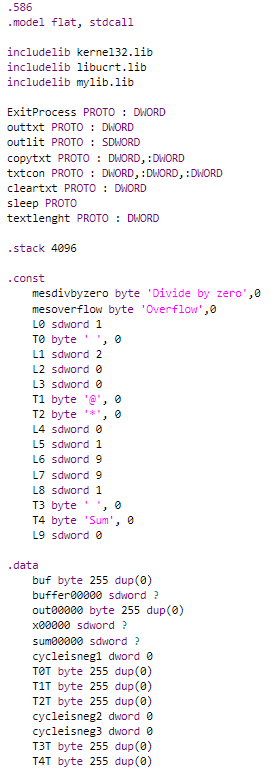
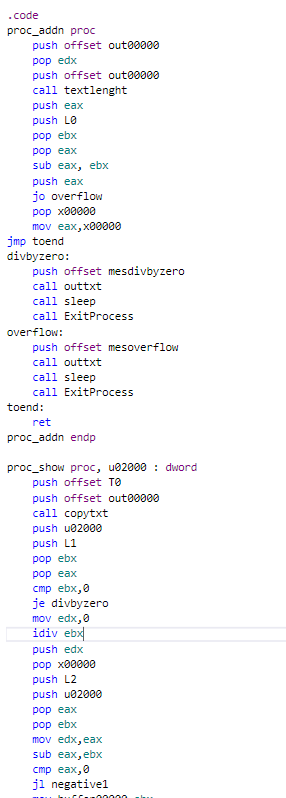
# **Приложение Д**

Дерево разбора

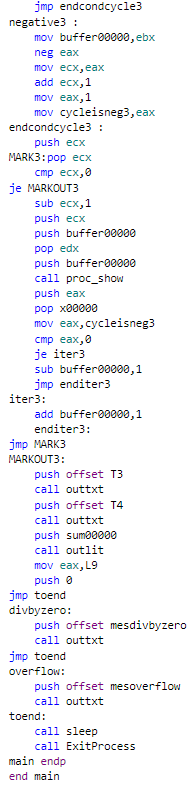
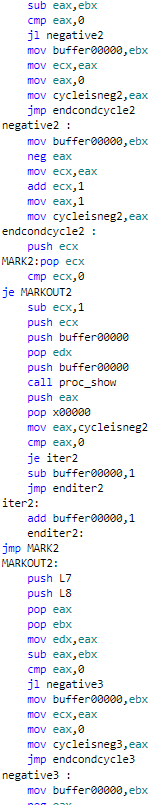
# **Приложение Е**

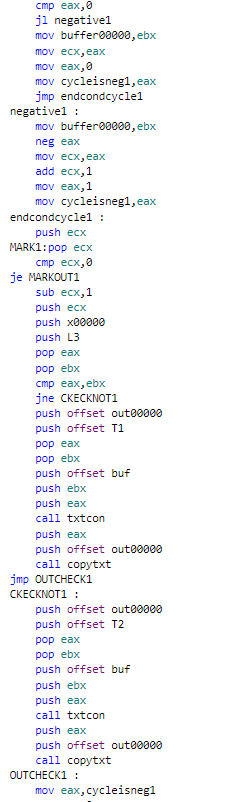
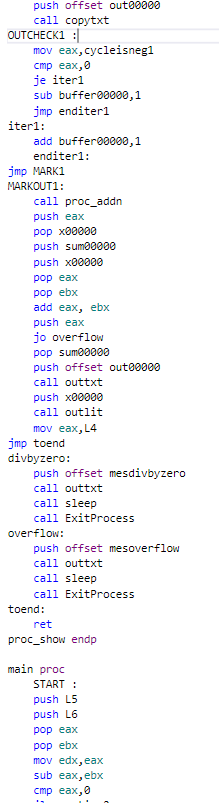
Ассемблерный код контрольного примера

# 

# 

# **Заключение**

* Реализованы 5 арифметических оператора для вычисления выражений;
* Реализована переменная по умолчанию
* Ревлизована поддержка глобальных переменных
* Реализованы два типа данных;
* Реализована конкатенация строк через оператор “+”;
* Поддерживается оператор вывода;
* Реализована библиотека для обработки строковых типов данных;
* Обрабатывается более 50 возможных ошибок;
* Выполнена генерация в ассемблер
* Реализован оператор цикла
* Реализован условный оператор