МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора BDA-2017»

Выполнил студент Будаев Дмитрий Александрович

(Ф.И.О.)

Руководитель проекта асп. Панченко О.Л.

(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 асп. Панченко О.Л.

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

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

Нормоконтролер асп. Панченко О.Л.

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

Курсовой проект защищен с оценкой

Минск 2017

Оглавление

[Глава 1. Спецификация языка программирования 2](#_Toc501592483)

[1.1 Характеристика языка программирования 2](#_Toc501592484)

[1.2 Алфавит языка 2](#_Toc501592485)

[1.3 Применяемые сепараторы 2](#_Toc501592486)

[1.4 Применяемые кодировки 3](#_Toc501592487)

[1.5 Типы данных 3](#_Toc501592488)

[1.6 Преобразование типов данных 3](#_Toc501592489)

[1.7 Идентификаторы 4](#_Toc501592490)

[1.8 Литералы 4](#_Toc501592491)

[1.9 Объявления данных и область видимости 4](#_Toc501592492)

[1.10 Инициализация данных 5](#_Toc501592493)

[1.11 Инструкции языка 5](#_Toc501592494)

[1.12 Операции языка 5](#_Toc501592495)

[1.13 Выражения и их вычисления 6](#_Toc501592496)

[1.14 Программные конструкции языка 6](#_Toc501592497)

[1.15 Область видимости идентификаторов 6](#_Toc501592498)

[1.16 Семантические проверки 6](#_Toc501592499)

[1.17 Распределение оперативной памяти на этапе выполнения 7](#_Toc501592500)

[1.18 Стандартная библиотека и её состав 7](#_Toc501592501)

[1.19 Ввод и вывод данных 7](#_Toc501592502)

[1.20 Точка входа 8](#_Toc501592503)

[1.21 Препроцессор 8](#_Toc501592504)

[1.22 Соглашения о вызовах 8](#_Toc501592505)

[1.23 Объектный код 8](#_Toc501592506)

[1.24 Классификация сообщений транслятора 8](#_Toc501592507)

[1.25 Контрольный пример 9](#_Toc501592508)

[Глава 2. Структура транслятора 10](#_Toc501592509)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 10](#_Toc501592510)

[2.2 Перечень входных параметров транслятора 11](#_Toc501592511)

[2.3 Перечень протоколов, формируемых транслятором и их содержимое 11](#_Toc501592512)

[Глава 3. Разработка лексического анализатора 12](#_Toc501592513)

[3.1 Структура лексического анализатора 12](#_Toc501592514)

[3.2 Контроль входных символов 12](#_Toc501592515)

[3.3 Удаление избыточных символов 13](#_Toc501592516)

[3.4 Перечень ключевых слов, сепараторов, символов операций соответствующих им лексем 14](#_Toc501592517)

[3.6 Принцип обработки ошибок 15](#_Toc501592518)

[3.7 Структура и перечень сообщений лексического анализатора 15](#_Toc501592519)

[Глава 4. Разработка синтаксического анализатора 16](#_Toc501592520)

[4.1 Структура синтаксического анализатора. 16](#_Toc501592521)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 16](#_Toc501592522)

[4.3 Построение конченого магазинного автомата 17](#_Toc501592523)

[4.4 Основные структуры данных 18](#_Toc501592524)

[4.5 Описание алгоритма синтаксического разбора 18](#_Toc501592525)

[4.6 Структура и перечень сообщений синтаксического анализатора 18](#_Toc501592526)

[4.7 Параметры синтаксического анализатора и режимы его работы 19](#_Toc501592527)

[4.8 Принцип обработки ошибок 19](#_Toc501592528)

[4.9 Контрольный пример 19](#_Toc501592529)

[Глава 5. Разработка семантического анализатора 20](#_Toc501592530)

[5.1 Структура семантического анализатора 20](#_Toc501592532)

[5.2 Функции семантического анализатора 20](#_Toc501592533)

[5.3 Структура и перечень сообщений семантического анализатора 21](#_Toc501592534)

[5.4 Принцип обработки ошибок 21](#_Toc501592535)

[5.5 Контрольный пример 21](#_Toc501592536)

[Глава 6. Преобразование выражений 22](#_Toc501592537)

[6.1 Выражения, допускаемые языком 22](#_Toc501592538)

[6.2 Польская запись и принцип ее построения 22](#_Toc501592539)

[6.3 Программная реализация обработки выражений 23](#_Toc501592540)

[6.4 Контрольный пример 23](#_Toc501592541)

[Глава 7. Генерация кода 24](#_Toc501592542)

[7.1 Структура генератора кода 24](#_Toc501592543)

[7.2 Представление типов данных в памяти 24](#_Toc501592544)

[7.3 Статическая библиотека 25](#_Toc501592545)

[7.4 Особенности алгоритма генерации кода 26](#_Toc501592546)

[7.5 Контрольный пример 26](#_Toc501592547)

[Глава 8. Тестирование транслятора 27](#_Toc501592548)

[Заключение 30](#_Toc501592549)

[Приложение А 31](#_Toc501592550)

[Приложение Б 32](#_Toc501592551)

[Приложение Г 34](#_Toc501592552)

[Приложение Д 36](#_Toc501592553)

[Приложение Е 37](#_Toc501592554)

[Литература 40](#_Toc501592555)

**Введение**

Транслятор – это комплекс отдельных программ, позволяющих преобразовывать исходный код на одном языке программирования в исходный код на другом языке программирования.

Классический транслятор состоит из следующих частей:

1. Лексический анализатор;

2. Синтаксический анализатор;

3. Семантический анализатор:

4. Генератор кода, или интерпретатор.

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

# Глава 1. Спецификация языка программирования

* 1. **Характеристика языка программирования**

Язык программирования BDA-2017 является универсальным, строготипизируемым, не объектно-ориентированным, транслируемым.

* 1. **Алфавит языка**

Алфавит BDA-2017 основан на таблице символов ACSII представленной на рисунке 1.1



Рисунок 1.1 - Алфавит входных символов

* 1. **Применяемые сепараторы**

Сепараторы служат для разделения операций языка. Сепараторы, применяемые в языке программирования BDA-2017, приведены в таблице 1.1

Таблица 1.1 - Применяемые сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение сепаратора |
| ; | Разделитель инструкций |
| { } | Программный блок |
| ( ) | Параметры;  Приоритетность операций |
| ˽ | Допускается везде, кроме идентификаторов и ключевых слов. Служит для их разделения |
| , | Разделитель параметров в функции |

* 1. **Применяемые кодировки**

При трансляции применяется кодировка Windows – 1251. Описание кодировки представлено в пункте 1.1.

* 1. **Типы данных**

Допускается использование фундаментальных типов данных определенных в таблице 1.2. Пользовательские типы данных не поддерживаются.

Таблица 1.2 - Фундаментальные типы данных

|  |  |
| --- | --- |
| Тип данных | Описание |
| number | Является целочисленным типом данных(4 байта). Предназначен для арифметических операций над числами. Инициализация по умолчанию: 0.  Предусмотрены следующие арифметические операции:  + - бинарная операция суммирования(number+number)  - - бинарная операция вычитания(number-number)  \* - бинарная операция умножения(number\*number)  / - бинарная операция деления(number/number) |
| line | Является строковым типом данных. Используется для работы с символами, каждый символ занимает 1 байт. Максимальное количество символов – 255.  Инициализация по умолчанию: нулевой символ.  Операции над данными строкового типа: возможно присваивание строковому идентификатору значения другого строкового идентификатора, строкового литерала или значения строковой функции, конкатенация строк. |

* 1. **Преобразование типов данных**

Преобразование не поддерживается.

* 1. **Идентификаторы**

Идентификаторы могут выступать в качестве имен функций, имен параметров, имен переменных. Зарезервированные идентификаторы не предусмотрены.

Имя идентификатора составляется по следующим правилам:

- состоит из символов латинского алфавита [a..z].

- длина идентификатора не должна превышать 10 символов. При превышении максимально допустимой длины длина идентификатора усекается до 10.

* 1. **Литералы**

Предусмотрены числовые(number) и строковые(line) литералы. Правила записи приведены в таблице 1.3

Таблица 1.3 - Правила записи литералов

|  |  |
| --- | --- |
| Тип литерала | Описание литерала |
| Числовые(number) | Обязательно содержат в себе только цифры[0..9] без дробной части. Максимально допустимым значением является число . Минимально допустимым значением является . При выходе за пределы допустимости, выводится соответствующая ошибка. Предусмотрена инициализация отрицательными литералами. |
| Строковые(line) | Символы, заключенные в “…”(двойные кавычки), число которых не превышает 255. В случае превышения длины литерала работа транслятора прекращается. |

* 1. **Объявления данных и область видимости**

В языке программирования BDA-2017 необходимо объявить переменную до ее использования. Областью видимости переменной является блок функции, в которой она определена. Объявление глобальных переменных не допускается. Область видимости построена по принципу C++(сверху вниз). Для объявления переменных используется следующая конструкция:

var˽<тип данных>˽<идентификатор>;

* 1. **Инициализация данных**

При объявлении переменной допускается инициализация данных. При этом переменной будет присвоено значение литерала или идентификатора, стоящего справа от знака равенства. Объектами-инициализаторами могут быть идентификаторы, литералы, выражения и вызовы функций. При объявлении без инициализации предусмотрены значения по умолчанию: значение 0 для целочисленного типа данных и строка длины 0 ("") для строкового типа данных.

* 1. **Инструкции языка**

В языке программирования BDA-2017 применяются инструкции, представленные в таблице 1.4.

Таблица 1.4 - Инструкции языка

|  |  |
| --- | --- |
| Инструкция языка | Синтаксис |
| Объявление переменной | var˽<тип данных>˽<идентификатор>; |
| Объявление функции | <тип данных>˽function˽<идентификатор>{тело функции}; |
| Присваивание | line:  <идентификатор> = <литерал>;  <идентификатор> = <выражение>;  <идентификатор 1> = <идентификатор 2>;  numeric:  <идентификатор> = <литерал>;  <идентификатор> = <выражение>;  <идентификатор 1> = <идентификатор 2>; |
| Главная функция | main  {  <инструкции языка>  } |
| Вызов функций | <идентификатор функции>(<идентификатор, литерал>,…,<идентификатор, литерал>); |
| Вывод данных | bdaout <идентификатор, литерал>; |
| Возврат из функции | return <идентификатор>; |

* 1. **Операции языка**

Операция сложения применима как к строковым типам данных, так и целочисленным. В случае строковых типов данных происходит конкатенация строк. Остальные операции в языке программирования применимы исключительно к целочисленным типам данных и приведены в таблице 1.5.

Таблица 1.5 - Операции языка

|  |  |
| --- | --- |
| Тип | Операторы |

Продолжение таблицы 1.5

|  |  |
| --- | --- |
| Арифметические | () – приоритетность операций  + ­­­ ̶ ­­­сложение  - ̶ вычитание  \* ̶ умножение  / ̶ деление |

* 1. **Выражения и их вычисления**

Предусмотрены следующие правила составления выражений:

* + Выражения записываются строго в одну строку.
  + Рассматриваются слева направо.
  + Для изменения приоритета операция используются круглые скобки ()
  + Реализация выражений происходит с помощью обратной польской записи.
  1. **Программные конструкции языка**

Программные конструкции представлены в таблице 1.7

Таблица 1.7 - Программные конструкции языка

|  |  |
| --- | --- |
| Конструкция | Представление в языке |
| Главная функция | main  {  <инструкции языка>  } |
| Функция | <тип данных> function ˽ <идентификатор>(<тип данных>˽<идентификатор>,…,<тип данных>˽<идентификатор>)  {  <инструкции языка>  } |
| Точка входа | main |

* 1. **Область видимости идентификаторов**

Все идентификаторы являются локальными и обязаны быть объявленными внутри какой-либо функции. Глобальных переменных нет. Параметры видны только внутри функции, в которой объявлены.

* 1. **Семантические проверки**

Перечень семантических проверок, предусмотренных языком, приведен в таблице 1.8.

Таблица 1.8 - Перечень семантических проверок

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Наличие функции main – точки входа в программу. |
| 2 | Единственность точки входа |
| 3 | Переопределение идентификаторов |
| 4 | Использование идентификаторов без их объявления |
| 5 | Проверка ограничений на количество параметров объявляемых функций |
| 6 | Правильность передаваемых в функцию параметров: количество, типы |
| 7 | Правильность строковых выражений |
| 8 | Превышение размера строковых и числовых литералов |

* 1. **Распределение оперативной памяти на этапе выполнения**

Транслированный код использует две области памяти. В сегмент констант заносятся все литералы. В сегмент данных заносятся переменные и параметры функций. Локальная область видимости в исходном коде определяется за счет использования правил именования идентификаторов и регулируется их префиксами, что и обуславливает их локальность на уровне исходного кода.

* 1. **Стандартная библиотека и её состав**

В языке BDA-2017 присутствует стандартная библиотека, которая подключается автоматически при трансляции исходного кода в язык ассемблера. Содержимое библиотеки и описание функций представлено в таблице 1.9.

Таблица 1.9 - Стандартная библиотека языка BDA-2017

|  |  |
| --- | --- |
| Функция | Описание |
| number function squareroot(number a); | Целочисленная функция. Вычисляет и возвращает квадратный корень из числа a. |
| number function square(number a); | Целочисленная функция. Вычисляет и возвращает квадрат числа a. |
| line function sumstr(line a, line b); | Строковая функция. Возвращает результат конкатенации строк a, b, записанный в строку a. |

* 1. **Ввод и вывод данных**

Вывод данных осуществляется с помощью ключевого слова bdaout. В качестве аргумента может выступать как числовые, так и строковые идентификаторы, а так же различного рода выражения.

bdaout <идентификатор>;

bdaout <выражение>;

Ввод не предусмотрен.

* 1. **Точка входа**

Точкой входа в программе является ключевое слово “main”. Точка входа не может отсутствовать или быть переопределена.

* 1. **Препроцессор**

Препроцессор в языке BDA-2017 не предусмотрен.

* 1. **Соглашения о вызовах**

Используется соглашение cdecl, то есть все параметры передаются в стек справа налево, память освобождает вызывающий код.

* 1. **Объектный код**

Язык транслируется в язык ассемблера.

* 1. **Классификация сообщений транслятора**

В соответствии с префиксами будут различаться сообщения, представленные в таблице 1.10.

Таблица 1.10 - Классификация сообщений транслятора

|  |  |
| --- | --- |
| Префикс ошибки | Описание ошибки |
| [LEX]### | ### - код ошибки. Сообщение, генерируемое на этапе лексического анализа. |
| [SYN]### | ### - код ошибки. Сообщение, генерируемое на этапе синтаксического анализа. |

Продолжение таблицы 1.10

|  |  |
| --- | --- |
| Префикс ошибки | Описание ошибки |
| [SEM]### | ### - код ошибки. Сообщение, генерируемое на этапе семантического анализа. |
| [SYSTEM]### | ### - код ошибки. Сообщение, генерируемое при критической ошибке. |

* 1. **Контрольный пример**

Контрольный пример, написанный на языке BDA-2017, представлен в приложении А.

**Глава 2. Структура транслятора**

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

Входными данными для транслятора является исходный код, написанный на языке программирования BDA-2017.

Выходными данными является объектный код, а также протоколы работы транслятора, описанные в пункте 2.3.

Компоненты транслятора приведены на рисунке 2.1.



Рис 2.1 Структура транслятора BDA-2017

Изначально исходный код подается на вход лексическому анализатору, который проверяет исходный текст на недопустимые символы, выделяет ключевые слова, идентификаторы и литералы, а также формирует таблицы лексем и идентификаторов.

Следующей фазой транслятора является синтаксический анализатор, на вход которому подается таблица лексем, полученная на этапе лексического анализа. При синтаксически правильном построении программы осуществляется переход к следующему этапу трансляции, иначе, работа транслятора останавливается.

Семантический анализатор представляет собой набор функций, осуществляющих проверку на разных этапах работы транслятора правил, описанных с помощью эвристических алгоритмов. В зависимости от критичности ошибок, возникающих на данном этапе, работа транслятора может быть либо прекращена, либо продолжена.

Генерация кода осуществляется посредством чистой интерпретации, то есть не создаётся промежуточное представление кода. Генерация кода является заключительным этапом, во время исполнения которого формируется объектный код.

**2.2 Перечень входных параметров транслятора**

В таблице 2.1 представлены входные параметры, которые могут использоваться для представления работы транслятора.

Таблица 2.1 - Входные параметры транслятора

|  |  |  |
| --- | --- | --- |
| Параметр | Назначение | Тип |
| -in: | Указывает на файл с исходным кодом. Исходный код содержится в файле с расширением \*.txt | Обязательный |
| -out: | Указывает имя протокола. Если не указан явно, то имя протокола не формируется. | Не обязательный |
| -log: | Указывает имя протокола. Если не указан явно, то имя протокола формируется конкатенацией имени файла исходного кода и постфикса «.log» | Не обязательный |

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

В ходе работы транслятор формирует протокол, в соответствии с заданными входными параметрами. -log:<путь к файлу> - содержит информацию о входных параметрах, о количестве символов исходного кода, а так же таблицу лексем, таблицу идентификаторов, таблицу работы синтаксического анализатора, дерево разбора.

# 

# Глава 3. Разработка лексического анализатора

**3.1 Структура лексического анализатора**

Структура лексического анализатора представлена на рисунке 3.1

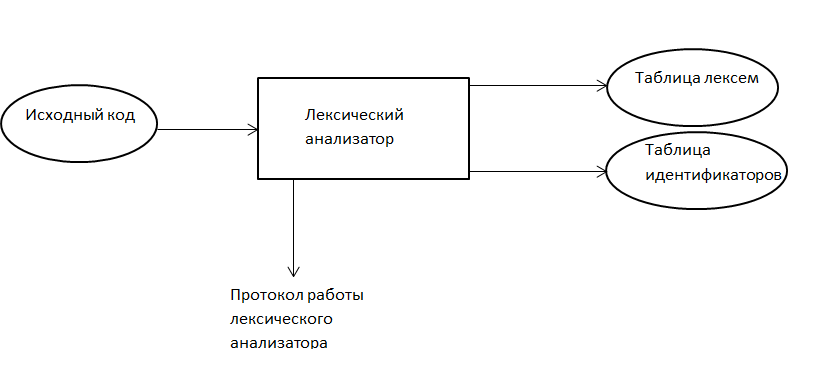


Рис 3.1 - Структура лексического анализатора

Входные данные: исходный код на языке BDA-2017;

Выходные данные: таблица лексем, таблица идентификаторов;

**3.2 Контроль входных символов**

Таблица допустимости представлена на рисунке 3.2.

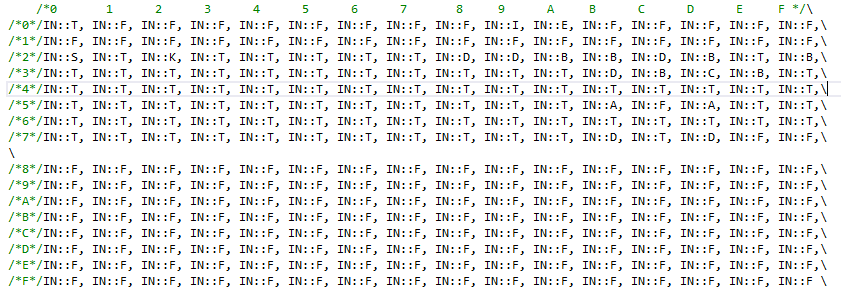


Рисунок 3.2 - Таблица допустимости входных символов

Таблица допустимости сформирована на основе кодировки windows-1251. Она предназначена для проверки входных символов на допустимость. Символы представлены в шестнадцатеричной системе счисления. Каждому символу в данной таблице соответствует некое числовое значение: «E»,«T»,«F»,«I»,«S»,«D»,«K»,«C»,«B».

«E» - символ окончания строки;

«T» - разрешенные алфавитом символы (также разрешенными являются все символы кроме «F» и «I»);

«F» - запрещенные алфавитом символы;

«I» - символы, которые игнорируются;

«S» - символ пробела;

«D» - символы, являющиеся сепараторами;

«K» - символы кавычек;

«C» - символ знака равно;

«B» - символы бинарных операторов;

**3.3 Удаление избыточных символов**

Избыточными символами на данном этапе являются пробелы, символы табуляции, символы перехода на новую строку.

Алгоритм удаления избыточных символов:

1. Просматриваем текущий символ
   1. Если «K», то записываем слово, пока не встретим закрывающую кавычку и заносим это слово в массив из токенов.
   2. Если «T», то записываем символ в результирующее слово.

1.2.1) Просматриваем следующий символ

1.2.1.1) Если «D»,«B»,«C»,«S»,«E», то заносим слово в массив токенов.

1.3) Если «B», «C», «D», то заносим слова в массив токенов.

1.4) Если «E», то инкрементируем переменную, отвечающую за подсчет строк.

1.5) Если «\0», то переход к пункту 3.

2) Перемещаем указатель на байт вправо и переходим к пункту 1.

3) Окончание алгоритма.

## 3.4 Перечень ключевых слов, сепараторов, символов операций соответствующих им лексем

Перечень ключевых слов, сепараторов, символов операций соответствующих им лексем представлен в таблице 3.1

Таблица 3.1 - Перечень ключевых слов

|  |  |
| --- | --- |
| Цепочка | Лексема |
| number | n |
| line | s |
| function | f |
| bdaout | p |
| return | r |
| main | m |
| var | d |
| + | + |
| - | - |
| \* | \* |
| / | / |
| = | = |
| ( | ( |
| ) | ) |
| { | { |
| } | } |
| , | , |
| ; | ; |
| идентификатор | i |
| числовой литерал | z |
| строковый литерал | l |

**3.5 Основные структуры данных**

Основные структуры данных, используемые на фазе синтаксического анализа, представлены в приложении Б.

**3.6 Принцип обработки ошибок**

При обнаружении критической ошибки, работа транслятора останавливается и происходит запись ошибки в log-файл. При некритической ошибке происходит только запись в log-файл. Подсчет количества ошибок не ведется.

## 3.7 Структура и перечень сообщений лексического анализатора

Префикс сообщений “LEX:”. Перечень сообщений, генерируемых на этапе лексического анализа, представлен в таблице 3.2

Таблица 3.2 - Сообщения лексического анализатора

|  |  |
| --- | --- |
| Код | Сообщение |
| 114 | Слишком длинный строковый литерал |
| 120 | Превышена максимальная длина идентификатора (>10). Усечение до 10 |
| 121 | Дублирование имени функции |
| 122 | Дублирование имени идентификатора или отсутствие области видимости |
| 123 | Неизвестная лексема |
| 124 | Неизвестный идентификатор |

**3.8 Параметры лексического анализатора и режим его работы**

Входным параметром является текст кода на языке BDA-2017. Параметры, определяющие режим работы лексического анализатора, не предусмотрены.

**3.9 Контрольный пример**

Результат работы лексического анализатора, на вход которого была подана программа на языке BDA-2017, описанная в пункте 1.25, представлен в приложении В.

# 

# Глава 4. Разработка синтаксического анализатора

## 4.1 Структура синтаксического анализатора.

Структура синтаксического анализатора представлена на рисунке 4.1.

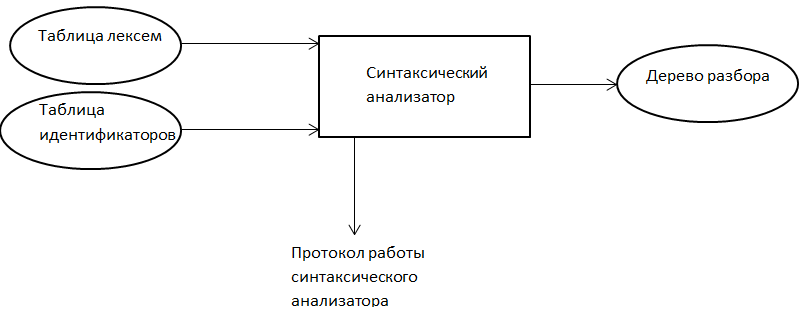


Рисунок 4.1 - структура синтаксического анализатора

Входные данные: таблица лексем, таблица идентификаторов.

Выходные данные: дерево разбора.

## 4.2 Контекстно-свободная грамматика, описывающая синтаксис языка

Грамматика, описывающая язык BDA-2017 представлена в таблице 4.1.

Таблица 4.1 - грамматика языка BDA-2017

|  |  |
| --- | --- |
| 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 |

## 4.3 Построение конченого 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 и правила перехода, используемые на фазе синтаксического анализа, представлены в приложении Г.

## 4.5 Описание алгоритма синтаксического разбора

На входе конченого автомата имеется лента, на которой находятся входные символы, а также грамматика в форме Грейбах.

1) если лента не пустая, переход к п.2, иначе переход к п.5

2) если на верхушке магазина нетерминальный символ

2.1) если есть такое правило

2.1.1) если есть цепочка, возвращаем NS\_OK. Переход к п.4.

2.1.2) иначе восстанавливаем состояние. Переход к п.4.

2.2) иначе возвращаем ошибку. Переход к п.4.

3) если на верхушке терминал и он совпадает с символом на ленте, то выталкиваем его из стека и продвигаем ленту. Переход к п.4.

4) Повторить шаг, переход к п.1.

5) Окончание работы

## 4.6 Структура и перечень сообщений синтаксического анализатора

Префикс сообщений “[SYN]:”. Перечень сообщений, генерируемых на этапе синтаксического анализа, представлен в таблице 4.3

Таблица 4.3 - Перечень сообщений

|  |  |
| --- | --- |
| Код | Сообщение |
| 200 | Неверная структура программы |
| 201 | Ошибочный оператор |
| 202 | Ошибка в выражении |
| 203 | Ошибка в расстановке знаков |
| 204 | Ошибка в формальных параметрах |
| 205 | Ошибка в параметрах вызываемой функции |

## 4.7 Параметры синтаксического анализатора и режимы его работы

Входными параметрами для синтаксического анализатора являются таблица лексем и таблица идентификаторов, полученные в ходе лексического анализа. По окончании разбора формируется дерево разбора, которое выводится в протокол работы –log:

## 4.8 Принцип обработки ошибок

При обнаружении ошибки в цепочке какого либо правила, СА проходит вверх по дереву разбора, пока не найдет верный вариант. Иначе запоминается самая глубокая ошибка, которая выводится в протокол работы.

## 4.9 Контрольный пример

Результатом работы синтаксического анализатора является дерево разбора, содержимое которого представлено в приложении Д.

# Глава 5. Разработка семантического анализатора

1. 1. **Структура семантического анализатора**

Семантический анализатор представляет собой набор функций, строящихся на эвристических алгоритмах, для проверки правильности исходной программы. Выполнение функций анализатора происходит на различных этапах работы транслятора. Структура семантического анализатора представлена на рисунке 5.1.

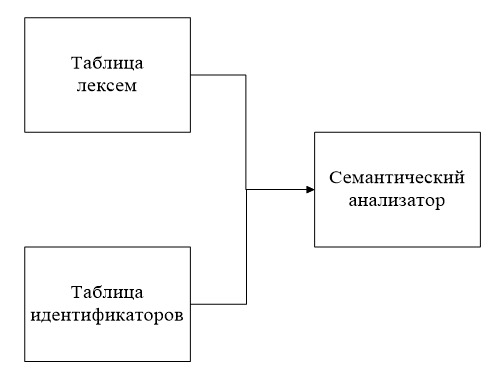
-

Рисунок 5.1 - Структура семантического анализатора

* 1. **Функции семантического анализатора**

Функции, представляющие проверку правил, представлены в таблице 5.1. Некоторые проверки встроены непосредственно в код этапов транслятора.

Таблица 5.1 - Семантические проверки

|  |  |
| --- | --- |
| Функция | Описание |
| Typecheck | Проверка, на соответствие типов формальных и фактических параметров |
| FuncRet | Проверка на возврат функции нужного типа данных |
| DoNotChangeParam | Проверка неизменности передаваемых параметров |
| ParamCheck | Проверка, на соответствие количества формальных и фактических параметров |

* 1. **Структура и перечень сообщений семантического анализатора**

Префикс сообщений “[SEM]:”. Сообщения, генерируемые при выполнении семантических проверок, представлены в таблице 5.2.

Таблица 5.2 - Перечень сообщений

|  |  |
| --- | --- |
| Код | Сообщение |
| 125 | Отсутствует точка входа main |
| 126 | Дублирование точки входа main |
| 300 | Несовпадение типов параметров функции |
| 301 | Несовпадение количества параметров функции |
| 302 | Присвоение аргументу не соответствующий тип |
| 303 | Недопустимые операции со строками |
| 304 | Операция над разными типами |
| 305 | Функция возвращает неверный тип |
| 306 | Вызов не существующей функции |
| 307 | Число выходит за границы допустимого |
| 308 | Нельзя изменять строковые параметры функции |

* 1. **Принцип обработки ошибок**

Ошибки, обнаруженные в ходе семантических проверок, являются критическими и приводят к прекращению работы транслятора. В случае обнаружения ошибки, работа транслятора прекращается, сообщение об ошибке выводится в протокол работы транслятора.

* 1. **Контрольный пример**

Результат работы семантических функций описан в главе 8.

**Глава 6. Преобразование выражений**

**6.1 Выражения, допускаемые языком**

В языке BDA-2017 допускаются выражения с использованием числовых идентификаторов и литералов. Также предусмотрены следующие арифметические операции:

- сложения: «+» ;

- вычитания: «-» ;

- умножения: «\*» ;

- деления: «/» ;

Также допустима возможность изменения приоритета выполнения арифметических операций при помощи скобок :

- «(»;

- «)»;

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 |
| «(» , «)» | 1 |
| «+» «-» | 2 |
| «\*» «/» | 3 |

**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. Известен следующий принцип построения польской записи:

* исходная строка: выражение;
* результирующая строка: польская запись;
* стек: пустой;
* исходная строка просматривается слева направо;
* операнды переносятся в результирующую строку;
* операция записывается в стек, если стек пуст;
* операция выталкивает все операции с большим или равным приоритетом в результирующую строку;
* отрывающая скобка помещается в стек;
* закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются.

**6.3 Программная реализация обработки выражений**

Преобразование в польскую запись происходит после этапа лексического и синтаксического анализа, во время этапа генерации кода на язык ассемблера.

**6.4 Контрольный пример**

Контрольный пример разбора выражения содержится в таблице 6.2.

Таблица 6.2 - Разбор выражения

|  |  |  |
| --- | --- | --- |
| Выражение | Стек | Результат |
| ((x-3)\*y+1)/z |  |  |
| (x-3)\*y+1)/z | ( |  |
| x-3)\*y+1)/z | ( ( |  |
| -3)\*y+1)/z | ( ( | x |
| 3)\*y+1)/z | ( ( - | x |
| )\*y+1)/z | ( ( - | x 3 |
| \*y+1)/z | ( | x 3 - |
| y+1)/z | ( \* | x 3 - |
| +1)/z | ( \* | x 3 - y |
| 1)/z | ( + | x 3 - y \* |
| )/z | ( + | x 3 - y \* 1 |
| /z |  | x 3 - y \* 1 + |
| z | / | x 3 - y \* 1 + |
|  | / | x 3 - y \* 1 + z |
|  |  | x 3 - y \* 1 + z / |

# 

# Глава 7. Генерация кода

## 7.1 Структура генератора кода

В языке BDA-2017 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, который будет являться результатом работы транслятора. В случае возникновения ошибок генерация кода не будет осуществляться. Структура генератора кода BDA-2017 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 BDA-2017 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка BDA-2017 и языка ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке BDA-2017 | Тип идентификатора на языке ассемблера | Пояснение |
| number | SDWORD | Хранит целочисленный тип данных. |
| line | BYTE | Хранит указатель на начало строки. Строка должна завешаться нулевым символом. |

## 7.3 Статическая библиотека

В языке BDA-2017 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 манипулирования выводом, недоступные конечному пользователю. Эти функции представлены в таблице 1.9.

Таблица 1.9 - Дополнительные функции стандартной библиотеки

|  |  |
| --- | --- |
| Функция на языке С++ | Описание |
| void outint(int i) | Функции для вывода в стандартный поток значения целочисленного идентификатора/литерала. |
| void outstr(char\* s) | Функции для вывода в стандартный поток значения строкового идентификатора/литерала. |
| char\* \_ copystr(char\* str1, char\* str2) | Функция, копирующая значение строки str2 в строку str1. |
| char\* sumstr(char\* str1, char\* str2) | Строковая функция. Возвращает результат конкатенации строк a, b, записанный в строку a. |
| int square(int i) | Целочисленная функция. Вычисляет и возвращает квадратный корень из числа a. |
| int squareroot (int i) | Целочисленная функция. Вычисляет и возвращает квадрат числа a. |

## 7.4 Особенности алгоритма генерации кода

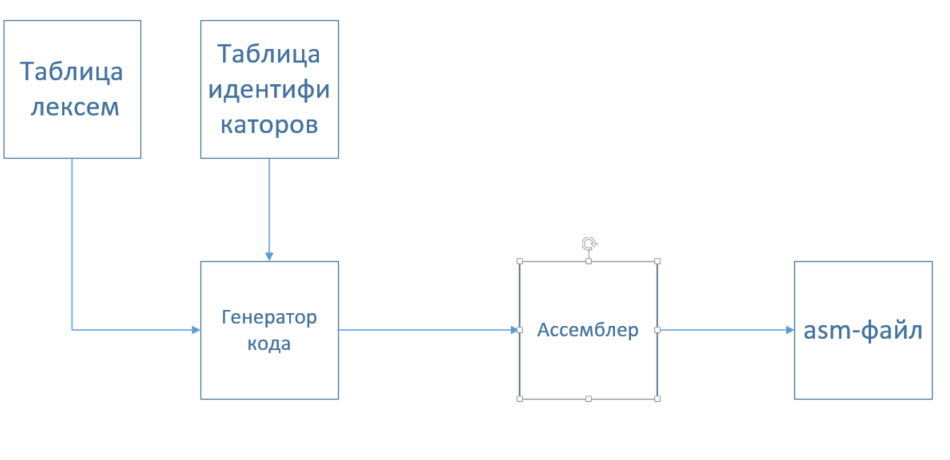
В языке BDA-2017 генерация кода строится на основе таблиц лексем и идентификаторов. Общая схема работы генератора кода представлена на рисунке 7.2

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.

# Глава 8. Тестирование транслятора

В данной главе описаны возможные ошибки, возникающие на различных этапах работы транслятора. Результат тестирования представлен в таблице 8.1.

Таблица 8.1 - Тестирование транслятора

|  |  |
| --- | --- |
| Исходный код | Генерируемая ошибка |
| main{  var line a="s;} | Ошибка 113: [SYN]Отсутствуют закрывающие кавычки. Строка: 2 |
| main{  var line abcdefghiqwerty="s";} | Ошибка 120: [LEX]Превышена максимальная длина идентификатора (>10). Усечение до 10. Строка: 2. |
| number function a()  {return 2};  number function a()  {return 2};  main{  var line a="s";} | Ошибка 121: [LEX]Дублирование имени функции. Строка: 3. |
| main{  var number a;  var line a="s";} | Ошибка 122: [LEX]Дублирование имени идентификатора или отсутствие области видимости. Строка: 3. |
| main{  var number a#;  } | Ошибка 123: [LEX]Неизвестная лексема. Строка: 2. |
| main{  b=2;  } | Ошибка 124: [LEX]Неизвестный идентификатор. Строка: 2. |
| {  b=2;  } | Ошибка 125: [SEM]Отсутствует точка входа main. |
| main{  var number b=2;  }  main  {} | Ошибка 126: [SEM]Дублирование точки входа main. |
| main{  bdaout 2; | Ошибка 200: [SYN]Неверная структура программы. |
| main{  bdaout 2;;  } | 201: строка 2, [SYN]Ошибочный оператор |
| main{  bdaout 2++;  } | 202: строка 2, [SYN]Ошибка в выражении |

Продолжение Таблицы 8.1

|  |  |
| --- | --- |
| Исходный код | Генерируемая ошибка |
| main{  bdaout 2+(2+3;  } | 203: строка 2, [SYN]Ошибка в расстановке знаков или скобок |
| number function a(var number a)  {return 2;};  main{  bdaout 2;  } | 204: строка 1, [SYN]Ошибка в формальных параметрах |
| number function a()  {return 2;};  main{  bdaout a(var);  } | 205: строка 4, [SYN]Ошибка в параметрах вызываемой функции |
| number function a(number a)  {return 2;};  main{  bdaout a("s");  } | Ошибка 300: [SEM]Несовпадение типов параметров функции. Строка: 4. |
| number function a()  {  return 2;  };  main{  bdaout a(2);  } | Ошибка 301: [SEM]Несовпадение количества параметров функции. Строка: 6. |
| main{  var line s=2;  } | Ошибка 302: [SEM]Присвоение аргументу не соответствующий тип. Строка: 2. |
| main{  bdaout "ss"/"s";  } | Ошибка 303: [SEM]Недопустимые операции со строками. Строка: 2. |
| main{  bdaout "ss"+2;  } | Ошибка 304: [SEM]Операция над разными типами. Строка: 2. |

Продолжение Таблицы 8.1

|  |  |
| --- | --- |
| Исходный код | Генерируемая ошибка |
| line function s()  {retrun 2;};  main{  bdaout "ss";  } | Ошибка 305: [SEM]Функция возвращает неверный тип. Строка: 2. |
| main{  var line s;  bdaout s(2);  } | Ошибка 306: [SEM]Вызов не существующей функции. Строка: 3. |
| main{  var number a;  a=9999999999999;  } | Ошибка 307: [SEM]Число выходит за границы допустимого. Строка: 3. |
| line function d(line a, line b)  {  a=b;  return a;  };  main{  var number a;  } | Ошибка 308: [SEM]Нельзя изменять строковые параметры функции. Строка: 3. |

# Заключение

* Реализованы 4 арифметических оператора для вычисления выражений, два типа данных, функции
* Реализована конкатенация строк через оператор “+”
* Присутствует подключаемая стандартная библиотека
* Обрабатывается 30 возможных исключительных ситуаций
* Поддерживается оператор вывода
* Среднее время выполнения генерации кода - 0.2 секунды
* Количество строк – около 3000

# Приложение А

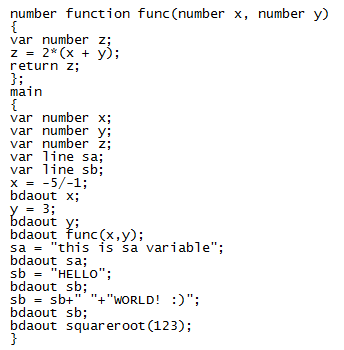
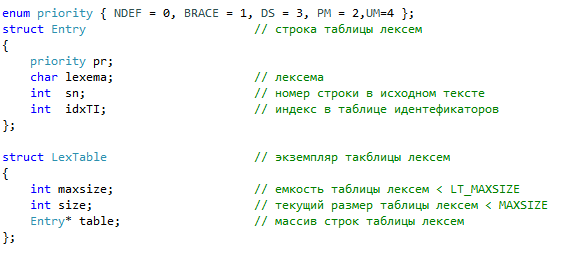
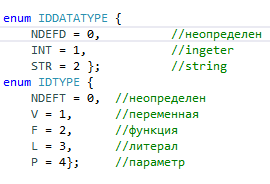


Рис. 8 Исходный код

# Приложение Б





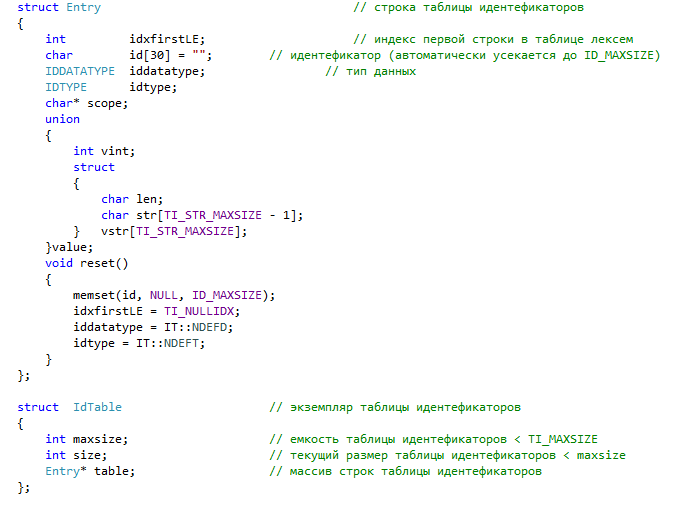


Рис. 9 Структуры данных, используемые на фазе лексического анализа

**Приложение В**

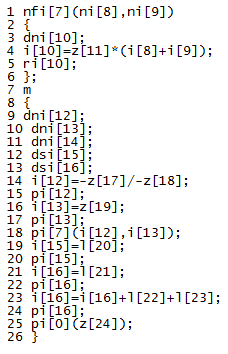


Рис. 10 Таблица лексем

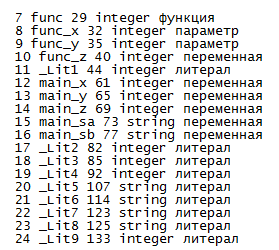
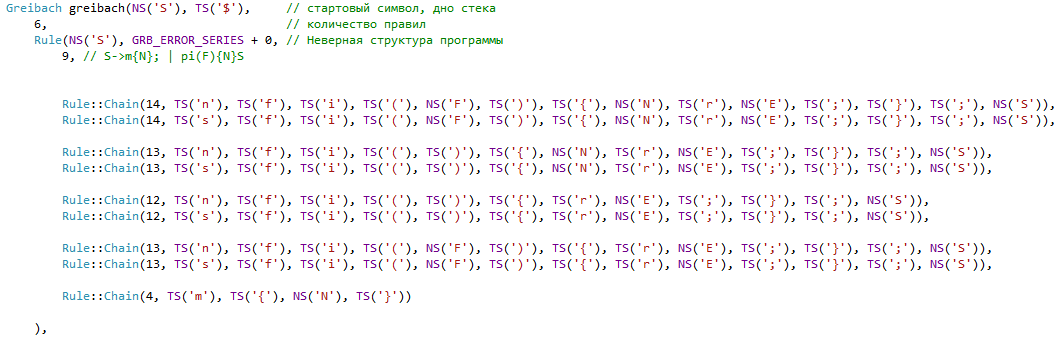


Рис. 11 Таблица идентификаторов

# Приложение Г



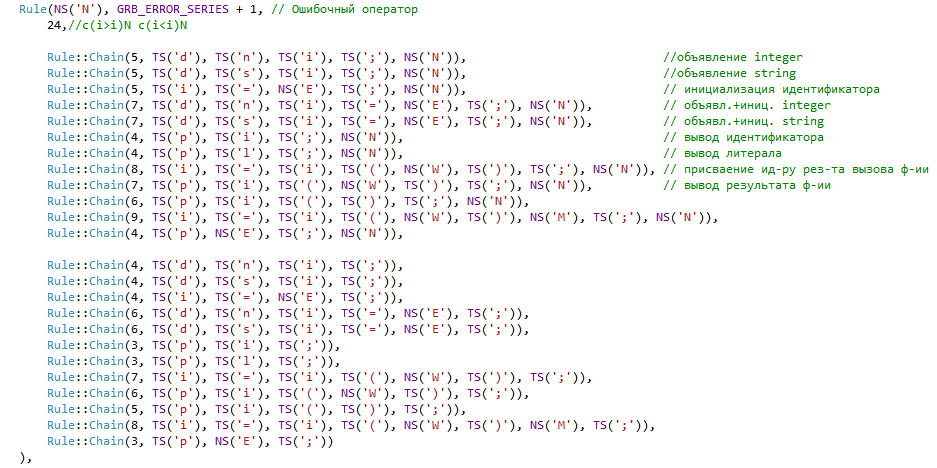
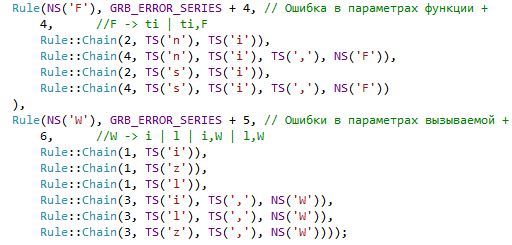


Рис.11 Структура данных грамматики Грейбах

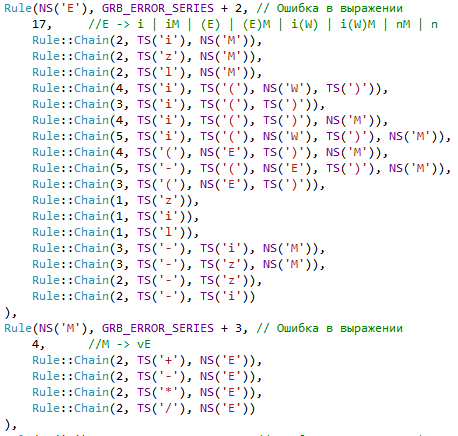


Рис. 11(прод) Структура данных грамматики Грейбах

# Приложение Д

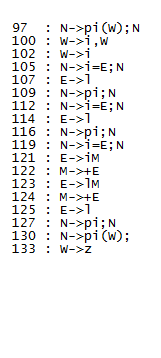
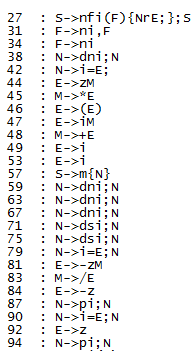


Рис. 12 Дерево разбора

## Приложение Е

Таблица 1 – Результат генерации кода

|  |
| --- |
| .586  .model flat, stdcall  includelib userlib.lib  includelib kernel32.lib  includelib libucrt.lib  ExitProcess PROTO : DWORD  squareroot PROTO : DWORD  copystr PROTO : DWORD, : DWORD  sumstr PROTO : DWORD, : DWORD  square PROTO : SDWORD  outstr PROTO : DWORD  outint PROTO : SDWORD  .stack 4096  .const  overflow db 'ERROR: VARIABLE OVERFLOW', 0  null\_division db 'ERROR: DIVISION BY ZERO', 0  \_Lit1 SDWORD 2  \_Lit2 SDWORD 5  \_Lit3 SDWORD 1  \_Lit4 SDWORD 3  \_Lit5 BYTE "this is sa variable", 0  \_Lit6 BYTE "HELLO", 0  \_Lit7 BYTE " ", 0  \_Lit8 BYTE "WORLD! :)", 0  \_Lit9 SDWORD 123  .data  func\_z SDWORD 0  main\_x SDWORD 0  main\_y SDWORD 0  main\_z SDWORD 0  main\_sa BYTE 255 DUP(0)  main\_sb BYTE 255 DUP(0)  .code  func\_proc PROC, func\_x : SDWORD, func\_y : SDWORD  push \_Lit1  push func\_x  push func\_y  pop eax  pop ebx  add eax, ebx  jo EXIT\_OVERFLOW  push eax  pop eax  pop ebx  imul eax, ebx  jo EXIT\_OVERFLOW  push eax  pop func\_z  push func\_z  jmp EXIT  EXIT\_DIV\_ON\_NULL:  push offset null\_division  call outstr  push - 1  call ExitProcess  EXIT\_OVERFLOW:  push offset overflow  call outstr  push - 2  call ExitProcess  EXIT:  pop eax  ret 8  func\_proc ENDP  main PROC  push \_Lit2  pop eax  neg eax  push eax  push \_Lit3  pop eax  neg eax  push eax  pop ebx  pop eax  test ebx,ebx  jz EXIT\_DIV\_ON\_NULL  cdq  idiv ebx  push eax  pop main\_x  push main\_x  call outint  push \_Lit4  pop main\_y  push main\_y  call outint  push main\_y  push main\_x  call func\_proc  push eax  call outint  push offset \_Lit5  push offset main\_sa  call copystr  push offset main\_sa  call outstr  push offset \_Lit6  push offset main\_sb  call copystr  push offset main\_sb  call outstr  push offset main\_sb  push offset \_Lit7  call sumstr  jo EXIT\_OVERFLOW  push eax  push offset \_Lit8  call sumstr  jo EXIT\_OVERFLOW  push eax  push offset main\_sb  call copystr  push offset main\_sb  call outstr  push \_Lit9  call squareroot  push eax  call outint  jmp EXIT  EXIT\_DIV\_ON\_NULL:  push offset null\_division  call outstr  push - 1  call ExitProcess  EXIT\_OVERFLOW:  push offset overflow  call outstr  push - 2  call ExitProcess  EXIT:  push 0  call ExitProcess  main ENDP  end main |

# Литература

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

2. Смелов, В.В. Курс лекций по предмету языки программирования – 2016

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

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