Содержание

[Глава 1. Спецификация языка программирования 5](#_Toc27412796)

[1.1 Характеристика языка программирования 5](#_Toc27412797)

[1.2 Алфавит языка 5](#_Toc27412798)

[1.3 Применяемые сепараторы 5](#_Toc27412799)

[1.4 Применяемые кодировки 6](#_Toc27412800)

[1.5 Типы данных 6](#_Toc27412801)

[1.6 Преобразование типов данных 6](#_Toc27412802)

[1.7 Идентификаторы 6](#_Toc27412803)

[1.8 Литералы 7](#_Toc27412804)

[1.9 Объявления данных и область видимости 7](#_Toc27412805)

[1.10 Инициализация данных 7](#_Toc27412806)

[1.11 Инструкции языка 8](#_Toc27412807)

[1.12 Операции языка 8](#_Toc27412808)

[1.13 Выражения и их вычисления 9](#_Toc27412809)

[1.14 Программные конструкции языка 9](#_Toc27412810)

[1.15 Область видимости идентификаторов 9](#_Toc27412811)

[1.16 Семантические проверки 9](#_Toc27412812)

[1.17 Распределение оперативной памяти на этапе выполнения 10](#_Toc27412813)

[1.18 Стандартная библиотека и её состав 10](#_Toc27412814)

[1.19 Ввод и вывод данных 11](#_Toc27412815)

[1.20 Точка входа 11](#_Toc27412816)

[1.21 Соглашения о вызовах 11](#_Toc27412817)

[1.22 Объектный код 11](#_Toc27412818)

[1.23 Классификация сообщений транслятора 11](#_Toc27412819)

[1.24 Контрольный пример 11](#_Toc27412820)

[Глава 2. Структура транслятора 12](#_Toc27412821)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 12](#_Toc27412822)

[2.2 Перечень входных параметров транслятора 13](#_Toc27412823)

[2.3 Перечень протоколов, формируемых транслятором и их содержимое 13](#_Toc27412824)

[Глава 3. Разработка лексического анализатора 14](#_Toc27412825)

[3.1 Структура лексического анализатора 14](#_Toc27412826)

[3.2 Контроль входных символов 14](#_Toc27412827)

[3.3 Удаление избыточных символов 15](#_Toc27412828)

[3.4 Перечень ключевых слов, сепараторов, символов операций соответствующих им лексем 15](#_Toc27412829)

[3.6 Принцип обработки ошибок 17](#_Toc27412830)

[3.7 Структура и перечень сообщений лексического анализатора 17](#_Toc27412831)

[Глава 4. Разработка синтаксического анализатора 18](#_Toc27412832)

[4.1 Структура синтаксического анализатора. 18](#_Toc27412833)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 18](#_Toc27412834)

[4.3 Построение конченого магазинного автомата 19](#_Toc27412835)

[4.4 Основные структуры данных 20](#_Toc27412836)

[4.5 Описание алгоритма синтаксического разбора 20](#_Toc27412837)

[4.6 Структура и перечень сообщений синтаксического анализатора 20](#_Toc27412838)

[4.7 Параметры синтаксического анализатора и режимы его работы 21](#_Toc27412839)

[4.8 Принцип обработки ошибок 21](#_Toc27412840)

[4.9 Контрольный пример 21](#_Toc27412841)

[Глава 5. Разработка семантического анализатора 22](#_Toc27412842)

[5.1 Структура семантического анализатора 22](#_Toc27412844)

[5.2 Функции семантического анализатора 22](#_Toc27412845)

[5.3 Структура и перечень сообщений семантического анализатора 22](#_Toc27412846)

[5.4 Принцип обработки ошибок 23](#_Toc27412847)

[5.5 Контрольный пример 23](#_Toc27412848)

[Глава 6. Преобразование выражений 24](#_Toc27412849)

[6.1 Выражения, допускаемые языком 24](#_Toc27412850)

[6.2 Польская запись и принцип ее построения 24](#_Toc27412851)

[6.3 Программная реализация обработки выражений 25](#_Toc27412852)

[6.4 Контрольный пример 25](#_Toc27412853)

[Глава 7. Генерация кода 26](#_Toc27412854)

[7.1 Структура генератора кода 26](#_Toc27412855)

[7.2 Представление типов данных в памяти 26](#_Toc27412856)

[7.3 Статическая библиотека 27](#_Toc27412857)

[7.4 Особенности алгоритма генерации кода 27](#_Toc27412858)

[7.5 Контрольный пример 27](#_Toc27412859)

[Глава 8. Тестирование транслятора 28](#_Toc27412860)

[Заключение 31](#_Toc27412861)

[Приложение А 33](#_Toc27412862)

[Приложение Б 34](#_Toc27412863)

[Приложение Г 36](#_Toc27412864)

[Приложение Д 39](#_Toc27412865)

[Приложение Е 40](#_Toc27412866)

[Литература 44](#_Toc27412867)

**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:

* 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.

# Глава 1. Спецификация языка программирования

* 1. **Характеристика языка программирования**

Язык программирования LVS-2019 является универсальным языком высокого уровня. Он является транслируемым, не объектно-ориентированным. Язык строго типизируемый, что говорит о невозможности преобразования типов, транслируемым языком программирования. В LVS-2019 используется два типа данных: целочисленные (int) и строковые (char) с функцией конкатенации и копирования строк.

* 1. **Алфавит языка**

В основе алфавита LVS-2019 лежит таблица символов ACSII. Недопустимо использование кириллицы, разрешены только латинские символы размером один байт, представленные на рисунке 1.1.

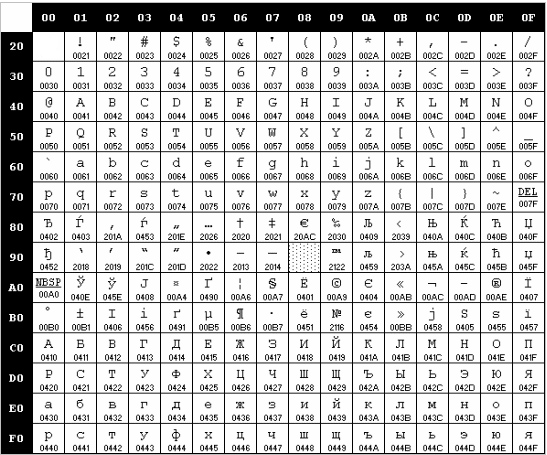


Рисунок 1.1 - Алфавит входных символов

* 1. **Применяемые сепараторы**

Сепараторы необходимы для разделения операций языка. Сепараторы, используемые в языке программирования LVS-2019, приведены в таблице 1.1.

1. Таблица 1.1 - Применяемые сепараторы

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

* 1. **Применяемые кодировки**

При трансляции применяется стандартная кодировка ACSII. Описание кодировки представлено в пункте 1.1.

* 1. **Типы данных**

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

Таблица 1.2 - Фундаментальные типы данных

|  |  |
| --- | --- |
| Тип данных | Описание |
| Int | Фундаментальный тип данных. Предусмотрен для объявления целочисленных положительных данных (4 байта). Диапазон от 0 до 4 296 967 296  Автоматически инициализируется нулевым значением.  Возможные операции:  + – бинарный, суммирование;  - – бинарный, вычитание;  \* – бинарный, умножение;  / – бинарный, деление;  Побитовые операции над числом:  || – бинарное, логическое или;  & – бинарный, логическое и;  ~ – унарный, инверсия;  = – присваивание значения; |
| Char | Фундаментальный тип данных. Предусмотрен для объявления символов. (1 символ – 1 байт).  Автоматически инициализируется нулем в соответствии с таблицей WINDOWS-1251 .  = – присваивание значения;. |

* 1. **Преобразование типов данных**

Преобразование не поддерживается, все типы данных определены однозначно и не могут быть преобразованы в другие, так как язык LVS-2019 является строго типизируемым.

* 1. **Идентификаторы**

Для именования функций, параметров и переменных используются Идентификаторы. Не предусмотрены зарезервированные идентификаторы. Имя идентификатора составляется по следующим образом:

* состоит из символов латинского алфавита [a..z].
* максимальная длина идентификатора равна 10 и не должна превышать это значение. При превышении максимально значения длина идентификатора усекается до 10.
  1. **Литералы**

Предусмотрены числовые (int) и строковые (char) литералы. Правила записи приведены в таблице 1.3.

Таблица 1.3 - Правила записи литералов

|  |  |
| --- | --- |
| Тип литерала | Описание литерала |
| Числовые | Максимально допустимое значение равно . Минимально допустимое значение равно . Могут состоять только из цифр [0..9] без дробной части. При выходе за пределы допустимости будет выведена соответствующая ошибка. |
| Строковые | Состоит из из символов, заключенных в "…" (двойные кавычки). Максимальное число которых не может превышать . В случае превышения длины литерала работа транслятора прекращается. |

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

В языке программирования LVS-2019 переменная должны быть объявлена до ее использования. Областью видимости переменной является блок функции, в которой она определена. Вне блока функции определение переменной запрещено. Недопустимо объявление глобальных переменных. Область видимости схожа с областью видимости C++, то есть сверху вниз. Конструкция для объявления переменных:

dec˽<типданных>˽<идентификатор>[=<литерал>|<идентификатор>| <выражение>];

* 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. Предусмотрены значения по умолчанию, если переменные не инициализированы: значение 0 для целочисленного и типа данных и строка длины 0 ("") для строкового типа данных.

* 1. **Инструкции языка**

В языке программирования LVS-2019 применяются инструкции, представ-ленные в таблице 1.4.

Таблица 1.4 - Инструкции языка

|  |  |
| --- | --- |
| Инструкция языка | Синтаксис |
| Главная функция | main  {  <инструкции языка>  } |
| Вызов функций | <идентификатор функции>(<идентификатор / литерал>,…) |
| Возврат из функции | return˽<идентификатор / литерал> |
| Объявление функции | <тип данных>˽function˽<идентификатор>  {  <инструкции языка>  } |
| Объявление переменной | dec˽<тип данных>˽<идентификатор> |
| Присваивание | char и int:  <идентификатор> = <литерал>;  <идентификатор> = <выражение>;  <идентификатор 1> = <идентификатор 2>; |
| Вывод данных | print˽<идентификатор / литерал> |

* 1. **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 LVS-2019 применимые к целочисленным и строковым типам данных приведены в таблице 1.5.

Таблица 1.5 - Операции языка

|  |  |
| --- | --- |
| Тип | Операторы |
| Арифметические | () – приоритетность операций  + ̶ сложение  -  ̶ вычитание  \* ̶ умножение  / ̶ деление |
| Битовые | ~, |, &, ^ |

* 1. **Выражения и их вычисления**

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.
  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>, …)  {  <инструкции языка>  } |
| Точка входа | main |

* 1. **Область видимости идентификаторов**

Все идентификаторы обязаны быть объявленными внутри функции. Вне функции объявление идентификаторов недопустимы. Глобальных переменных нет, только локальные. 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, как точки входа в программу |
| 2 | Наличие только одной точки входа |
| 3 | Переопределение идентификаторов |
| 4 | Использование идентификаторов без их объявления |
| 5 | Проверка на количество параметров объявляемых функций и ограничения применяемые к ним |
| 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 LVS-2019 присутствует стандартная библиотека, которая автоматически подключается при трансляции исходного кода в язык ассемблера. У каждого типа данных есть свои функции реализующие различные команды. Содержимое библиотеки и описание функций представлено в таблице 1.9.

Таблица 1.9 - Стандартная библиотека языка LVS-2019

|  |  |
| --- | --- |
| Функция | Описание |
| Rand(max) | Целочисленная функция. Вычисляет и возвращает случайное число. |
| atoi(i) | Целочисленная функция. Вычисляет и возвращает номер символа по таблице WINDOWS -1251 |

* 1. **Ввод и вывод данных**

Ввод данных не предусмотрен. Вывод данных осуществляется с помощью ключевого слова print. В качестве аргумента принимаются числовые и строковые идентификаторы, так же выражения:

print <идентификатор>;

print <выражение>;

* 1. **Точка входа**

В языке LVS-2019 точкой входа является ключевое слово “main”. Точка входа не может отсутствовать, также не может быть переопределена. В программе может быть только одна точка входа.

* 1. **Соглашения о вызовах**

При генерации кода используется соглашение \_stdcal ql, в котором все параметры передаются в стек справа налево. Освобождением памяти занимается вызываемая подпрограмма, которая очищает стек.

* 1. **Объектный код**

Исходный код языка транслируется в язык ассемблера.

* 1. **Классификация сообщений транслятора**

Транслятор генерирует сообщения о ошибках пользователю. 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. |
| 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 LVS-2019, представлен в приложении А.

**Глава 2. Структура транслятора**

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

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

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

Компоненты транслятора приведены на рисунке 2.1.



Рис 2.1 - Структура транслятора LVS-2019

Первоначально на вход лексического анализатора передается исходный код. Анализатором проверяется исходный текст на недопустимые символы, выделяет литералы, идентификаторы и ключевые слова, а также формирует таблицы лексем и идентификаторов.

Далее наступает черед синтаксического анализатора, к нему на вход поступа-ет таблица лексем, полученная на этапе лексического анализа. Если программа по-строена синтаксически правильно, то осуществляется переход к этапу трансляции стоящему далее, при ином раскладе работа транслятора останавливается.

Наборы функций, проверяющие правила на разных этапах работы транслято-ра представлены в семантическом анализаторе. Продолжение или остановка работы транслятора всецело зависит от критичности возникающих ошибок.

Генерация кода реализуется посредством чистой интерпретации, без создания промежуточного представления кода. В финале происходит генерация кода, 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**

По итогам своей работы транслятор формирует протокол, согласно заданным входным параметрам. -log: <путь к файлу> - в этом файле находятся информация о входных параметрах, количестве символов исходного кода, таблицы лексем, идентификаторов, 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

Исходный код на языке LVS-2019 является входными данными;

Таблицы лексем и идентификаторов являются выходными данными;

**3.2 Контроль входных символов**

Таблица допустимости представлена на рисунке 3.2.

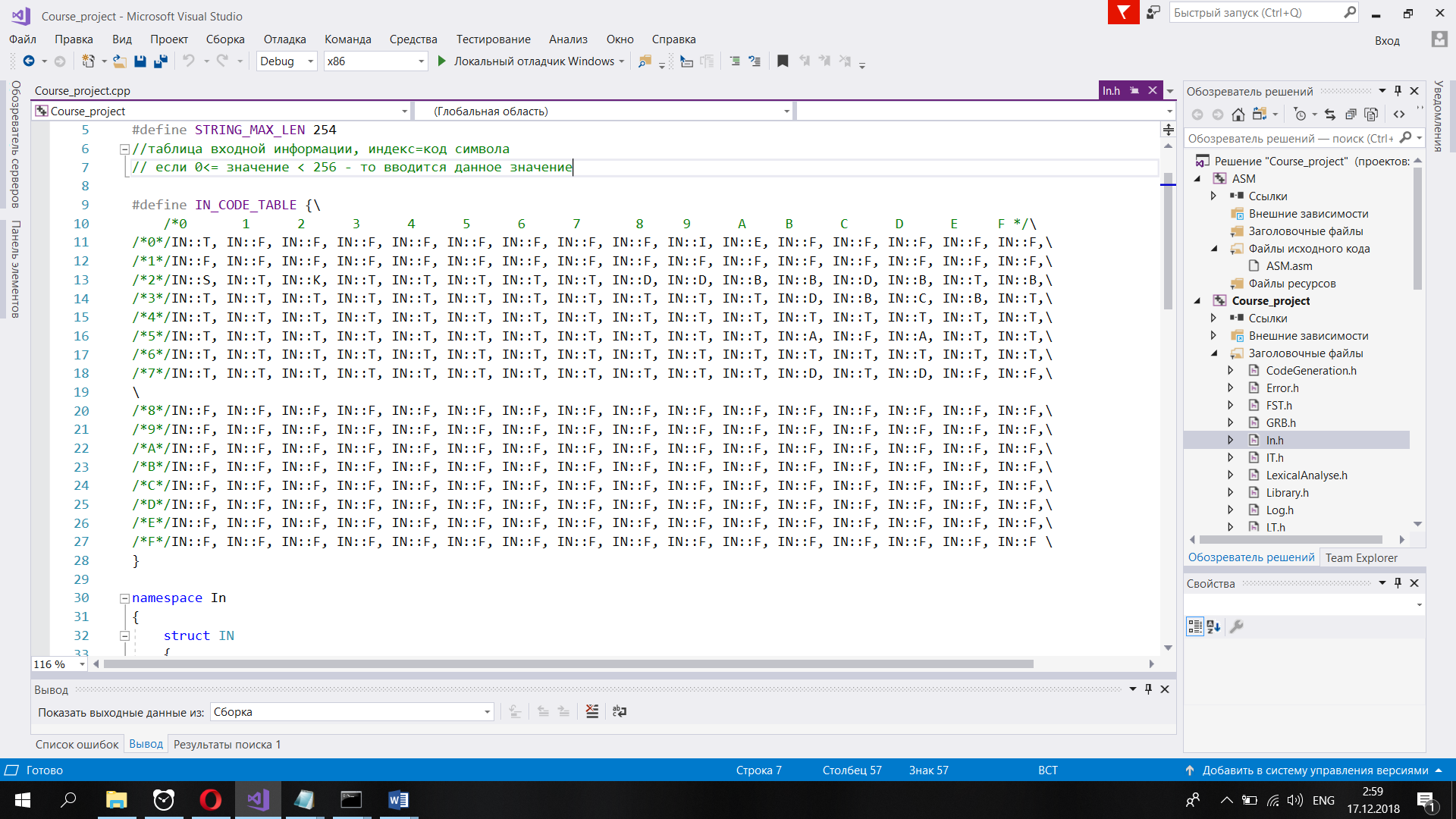


Рисунок 3.2 - Таблица допустимости входных символов

Таблица допустимости была сформирована на основе кодировки windows-1251. Таблица необходима для проверки входных символов на допустимость. Символы могут быть разрешенными, запрещенными, игнорируемыми и др.

Символы представлены в шестнадцатеричной системе счисления. В таблице записаны различные числовые значения соответствующие символам в данной таблице:

|  |  |
| --- | --- |
| «T» - разрешенные алфавитом символы;  «F» - запрещенные алфавитом символы;  «I» - символы, которые игнорируются;  «E» - символ окончания строки;  «D» - символы, являющиеся сепараторами;  «S» - символ пробела;  «K» - символы кавычек;  «C» - символ знака равно;  «B» - символы бинарных операторов; | SEP - ( ) \* + - = , / { } ;  PR - пробел  NS - новая строка  QUOTE - " "  F - запрещённый  T – разрешённый  I - игнорируемый |

**3.3 Удаление избыточных символов**

Избыточными символами представлены пробелы, символы табуляции, символы перехода на новую строку.

Алгоритм удаления избыточных символов:

1. Просматриваем текущий символ
   1. Если «K», символы кавычек, то записываем слово, пока не встретим закрывающую кавычку и заносим это слово в массив из токенов.
   2. Если «T», разрешенные символы, 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, то есть увеличиваем на единицу, отвечающую за подсчет строк.

1.5) Если «\0», нулевой символ, 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 |
| int | n |
| char | s |
| function | f |
| print | p |
| return | r |
| main | m |
| dec | d |
| + | + |
| - | - |
| \* | \* |
| / | / |
| = | = |
| ( | ( |
| ) | ) |
| { | { |
| } | } |
| , | , |
| ; | ; |
| ~ | ~ |
| ^ | ^ |
| | | | |
| & | & |
| идентификатор | i |
| числовой литерал | z |
| строковый литерал | l |

**3.5 Основные структуры данных**

В приложении Б представлены основные структуры данных на этапе синтаксического анализа.

**3.6 Принцип обработки ошибок**

В случае обнаружения критической ошибки, которая не позволяет работать анализаторам или генератору правильно функционировать, транслятор прекращает свою работу и в log-файл записывается ошибка. А при некритической ошибке происходит запись этой ошибки в log-файл без прекращения работы транслятора. Подсчет количества ошибок в обоих случаях 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 | Превышение длины строкового литерала |
| 120 | Превышение максимальной длины идентификатора. Литерал будет усечен до 10 |
| 121 | Имя функции было продублировано |
| 122 | Дублирование имени идентификатора или отсутствие области видимости |
| 123 | Неизвестная лексема |
| 124 | Неизвестный идентификатор |

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

Текст кода на языке LVS-2019 подается на вход. Параметры не определяют режим работы лексического анализатора.

**3.9 Контрольный пример**

На вход лексического анализатора подается программа на языке LVS-2019, описанная в пункте 1.25. Результат работы лексического анализатора 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

Таблицы лексем и идентификаторов являются входными данными.

Дерево разбора является выходными данными.

## 4.2 Контекстно-свободная грамматика, описывающая синтаксис языка

Грамматика, описывающая язык LVS-2019 представлена в таблице 4.1.

Таблица 4.1 - грамматика языка LVS-2019



## 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

В приложении Г представлены основные структуры данных и правила пере-хода, используемые на этапе синтаксического анализа.

## 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

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

Таблица 4.3 - Перечень сообщений

|  |  |
| --- | --- |
| Код | Сообщение |
| 200 | Неправильная 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

Таблицы идентификаторов и лексем являются входными параметрами для синтаксического анализатора. Эти таблицы мы получаем в ходе лексического анализа. В конце после разбора формируется дерево разбора, которое выводится в протокол работы –log.

## 4.8 Принцип обработки ошибок

При обнаружении ошибки в цепочке какого-либо правила, синтаксический анализатор идет 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. Результат работы синтаксического анализатора представлен в приложении Д.

# Глава 5. Разработка семантического анализатора

1. 1. **Структура семантического анализатора**

Семантический анализатор состоит из набора функций для проверки правильности исходной программы. Функции анализатора выполняются на различных этапах работы транслятора. Структура семантического анализатора представлена на рисунке 5.1.

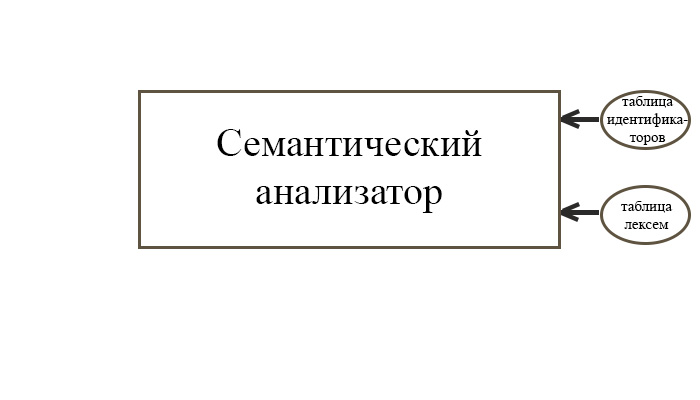


Рисунок 5.1 - Структура семантического анализатора

* 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 | Отсутствие точки входа main |
| 126 | Дублирование точки входа main |
| 300 | Несовпадение типов параметров функции |
| 301 | Несовпадение количества параметров функции |
| 302 | Присвоение аргументу не соответствующего типа |
| 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**

Все семантические ошибки являются критическими, из-за чего транслятор прекращает свою работу и в протокол работы транслятора выводится соответствующее сообщение об ошибке.

* 1. **Контрольный пример**

Результат работы семантических функций описан в главе 8.

**Глава 6. Преобразование выражений**

**6.1 Выражения, допускаемые языком**

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

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

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 |
| «(», «)» | 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**

После этапов лексического и синтаксического анализа происходит преобразование в польскую запись, 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 + x)\*k |  |  |
| x + x)\*k | ( |  |
| + x)\*k | ( | x |
| x)\*k | ( + | x |
| )\*k | ( + | x x |
| \*k |  | x x + |
| k | \* | x x + |
|  | \* | x x + k |
|  |  | x x + k \* |

# Глава 7. Генерация кода

## 7.1 Структура генератора кода

Заключительным этапом трансляции языка LVS-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 LVS-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 LVS-2019 и на языке ассемблера приведены в таблице 7.1.

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

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке LVS-2019 | Тип идентификатора на языке ассемблера | Пояснение |
| int | SDWORD | Хранит целочисленный тип данных. |
| char | BYTE | Хранит указатель на начало строки. Строка должна завешаться нулевым символом. |

## 7.3 Статическая библиотека

В языке LVS-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 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. |

## 7.4 Особенности алгоритма генерации кода

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

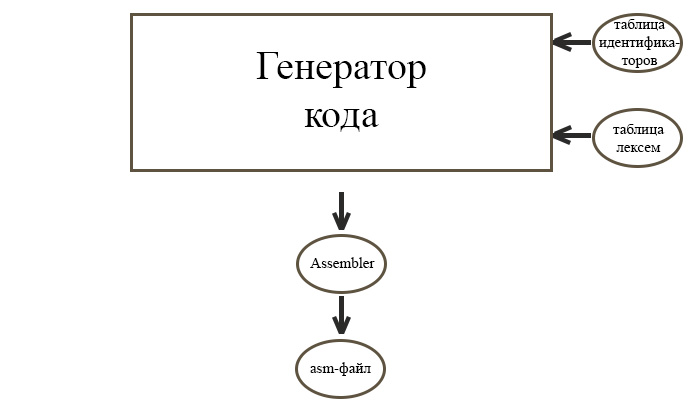


Рисунок 7.2 - Структура генератора кода

## 7.5 Контрольный пример

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

# 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  {  dec char a = "c;  } | Ошибка 113: SYN:Отсутствуют закрывающие кавычки. Строка: 3 |
| main  {  dec char abcdabcdabcd = "r";  } | Ошибка 120: LEX:Превышена максимальная длина идентификатора. Усечение до 10. Строка: 3. |
| int function а()  {  return 1;  };  int function а()  {  return 1;  };  main  {  dec char a = "char";} | Ошибка 121: LEX:Дублирование имени функции. Строка: 5. |
| main  {  dec int a;  dec int a = "char";  } | Ошибка 122: LEX:Дублирование имени идентификатора или отсутствие области видимости. Строка: 4. |
| main  {  dec int a#;  } | Ошибка 123: LEX:Неизвестная лексема. Строка: 3. |
| main  {  a = 1;  } | Ошибка 124: LEX:Неизвестный идентификатор. Строка: 3. |
| {  dec int a = 1;  } | Ошибка 125: SEM:Отсутствует точка входа “main”. |

Продолжение Таблицы 8.1

|  |  |
| --- | --- |
| Исходный код | Генерируемая ошибка |
| main  {  dec int a = 1;  }  main  {} | Ошибка 126: SEM:Дублирование точки входа “main”. |
| main  {  print 1; | Ошибка 200: SYN:Неверная структура программы. |
| main  {  print 1;;  } | 201: строка 3, SYN:Ошибочный оператор |
| main  {  print 1++;  } | 202: строка 3, SYN:Ошибка в выражении |
| main  {  print 1+(1+1;  } | 203: строка 3, SYN:Ошибка в расстановке знаков или скобок |
| int function a(int int a)  {  print 1;  };  main  {  print 1;  } | 204: строка 1, SYN:Ошибка в формальных параметрах |
| int function a()  {  return 1;  };  main  {  print a(dec);  } | 205: строка 7, SYN:Ошибка в параметрах вызываемой функции |

Продолжение Таблицы 8.1

|  |  |
| --- | --- |
| Исходный код | Генерируемая ошибка |
| int function a(int a)  {  return 1;  };  main  {  print a("char");  } | Ошибка 300: SEM:Несовпадение типов параметров функции. Строка: 7. |
| int function a()  {  return 1;  };  main  {  print a(1);  } | Ошибка 301: SEM:Несовпадение количества параметров функции. Строка: 6. |
| main  {  dec char a = 1;  } | Ошибка 302: SEM:Присвоение аргументу не соответствующий тип. Строка: 3. |
| main  {  char "char" / "char";  } | Ошибка 303: SEM:Недопустимые операции со строками. Строка: 3. |
| main  {  print "char" + 1;  } | Ошибка 304: SEM:Операция над разными типами. Строка: 3. |
| char function c()  {  return 1;  };  main  {  print "char";  } | Ошибка 305: SEM:Функция возвращает неверный тип. Строка: 3. |
| main  {  dec char c;  print с(1);  } | Ошибка 306: SEM:Вызов не существующей функции. Строка: 4. |

Продолжение Таблицы 8.1

|  |  |
| --- | --- |
| Исходный код | Генерируемая ошибка |
| main  {  dec int а;  a = 111111111111111;  } | Ошибка 307: SEM:Число выходит за границы допустимого. Строка: 4. |
| int function c(char a, char b)  {a = b;  print a;  };  main  {  dec int a;  } | Ошибка 308: SEM:Нельзя изменять строковые параметры функции. Строка: 3. |

Таким образом, при встрече ошибок в процессе работы транслятора программа выводит сообщение об ошибке в файл и завершает работу.

# 

# Заключение

* Реализованы 4 арифметических оператора для вычисления выражений;
* Реализованы два типа данных;
* Реализованы 4 битовых операции;
* поддерживается оператор вывода;
* Реализовано 3 функции для двух типов данных;
* присутствует подключаемая стандартная библиотека;

# 

# Приложение А

main()

{

dec int x = 15;

dec int y = 8;

dec int d = 0116;

dec int z = 0101;

dec int r;

dec int p;

dec int f;

dec int hf = 1;

dec char s;

dec char e = "e";

dec int i;

r = Rand(x);

i = atoi(e);

s = "a";

x = i;

x = 15;

z = (x+y)\*i;

p = x ^ y;

f = x | i;

print z;

print f;

print p;

print i;

print r;

print s;

return 0;

}

Рис. 8 - Исходный код

# Приложение Б

struct Entry

{

char lexema; // Лексема

int sn; // Номер строки в исходном тексте

int idxTI; // Индекс в таблице идентификаторов или LT\_TI\_NULLIDX

int tokenId = 0;

char operation;

std::char value;

Entry() {};

Entry(char lexema, int snn, int idxti = LT\_TI\_NULLDX);

Entry(char lexema, int snn, char v, int idxti = LT\_TI\_NULLDX);

};

struct LexTable // Экземпляр таблицы лексем

{

int maxsize; // Ёмкость таблицы лексем

int size; // Текущий размер таблицы лексем

Entry\* table; // Массив строк таблицы лексем

};

LexTable Create(int size); // Ёмкость < LT\_MAXSIZE

void Add(LexTable &lextable, Entry entry); // Экземпляр таблицы лексем, строка таблицы лексем

Entry GetEntry(LexTable &lextable, int n); // Экземпляр таблицы лексем, номер получаемой строки

void Delete(LexTable &lextable); // Удалить таблицу лексем (освободить память)

struct LIT { // Структура для записи литералов

std::char name[65]; //имя литерала

std::char literals[65]; //значение литерала

std::char typel[65]; //тип литерала

};Рис. 9 - Структуры данных, используемые на фазе лексического анализа

**Приложение В**

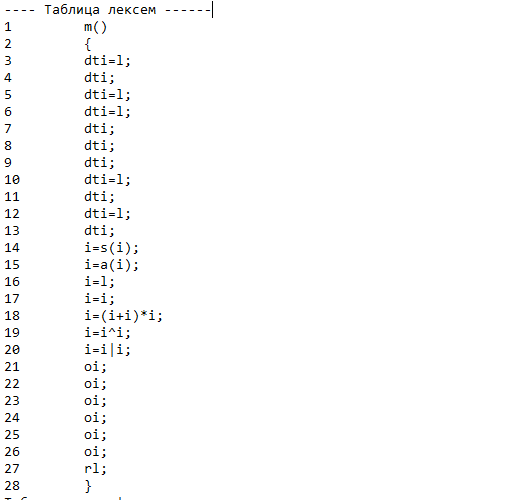


Рис. 10 - Таблица лексем

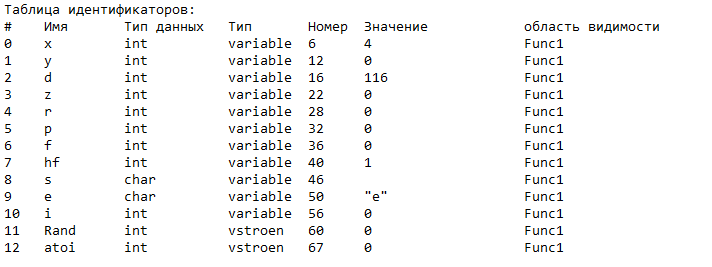


Рис. 11 - Таблица идентификаторов

# Приложение Г

Greibach greibach(NS('S'), TS('$'), 7,

Rule(NS('S'), GRB\_ERROR\_SERIES + 0, 6, // Структура программы

Rule::Chain(11, TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('}'), TS(';'), NS('S')),

Rule::Chain(8, TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS(';'), NS('S')),

Rule::Chain(10, TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('}'), NS('S')),

Rule::Chain(9, TS('t'), TS('f'), TS('i'), TS('('), TS(')'), TS('{'), NS('N'), TS('}'), NS('S')),

Rule::Chain(7, TS('m'), TS('('), TS(')'), TS('{'), NS('N'), TS('}'), TS(';')),

Rule::Chain(4, TS('m'), TS('{'), NS('N'), TS('}'))

),

Rule(NS('F'), GRB\_ERROR\_SERIES + 3, 2,// параметры ф-ии

Rule::Chain(2, TS('t'), TS('i')),

Rule::Chain(4, TS('t'), TS('i'), TS(','), NS('F'))

),

Rule(NS('N'), GRB\_ERROR\_SERIES + 4, 12, // возможные конструкции в ф-иях

Rule::Chain(5, TS('d'), TS('t'), TS('i'), TS(';'), NS('N')),

Rule::Chain(9, TS('d'), TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS(';'), NS('N')),

Rule::Chain(8, TS('d'), TS('t'), TS('p'), TS('('), NS('F'), TS(')'), TS(';'), NS('N')),

Rule::Chain(8, TS('d'), TS('t'), TS('e'), TS('('), NS('F'), TS(')'), TS(';'), NS('N')),

Rule::Chain(7, TS('d'), TS('t'), TS('i'), TS('='), NS('E'), TS(';'), NS('N')),

Rule::Chain(5, TS('i'), TS('='), NS('E'), TS(';'), NS('N')),

Rule::Chain(9, TS('u'), TS('('), NS('E'), TS(')'), TS('{'), NS('N'), TS('}'), TS(';'), NS('N')),

Rule::Chain(8, TS('u'), TS('('), NS('E'), TS(')'), TS('{'), NS('N'), TS('}'), TS(';')),

Rule::Chain(6, TS('i'), TS('='), NS('B'), NS('E'), TS(';'), NS('N')),

Rule::Chain(4, TS('o'), NS('E'), TS(';'), NS('N')),

Rule::Chain(4, TS('i'), TS('='), NS('E'), TS(';')),

Rule::Chain(3, TS('r'), NS('E'), TS(';'))

),

Rule(NS('E'), GRB\_ERROR\_SERIES + 2, 19, // выражения

Rule::Chain(1, TS('i')),

Rule::Chain(1, TS('l')),

Rule::Chain(3, TS('i'), TS('n'), TS('l')),

Rule::Chain(3, TS('i'), TS('n'), TS('i')),

Rule::Chain(3, TS('i'), TS('='), TS('l')),

Rule::Chain(2, TS('i'), NS('E')),

Rule::Chain(2, TS('i'), NS('M')),

Rule::Chain(2, TS('l'), NS('M')),

Rule::Chain(3, TS('('), NS('W'), TS(')')),

Rule::Chain(3, TS('('), NS('E'), TS(')')),

Rule::Chain(4, TS('('), NS('E'), TS(')'), NS('M')),

Rule::Chain(4, TS('s'), TS('('), NS('W'), TS(')')),

Rule::Chain(4, TS('a'), TS('('), NS('W'), TS(')')),

Rule::Chain(4, TS('p'), TS('('), NS('W'), TS(')')),

Rule::Chain(5, TS('a'), TS('('), NS('W'), TS(')'), NS('M')),

Rule::Chain(5, TS('s'), TS('('), NS('W'), TS(')'), NS('M')),

Rule::Chain(4, TS('p'), TS('('), NS('W'), TS(')'), NS('M')),

Rule::Chain(4, TS('('), NS('W'), TS(')'), NS('M')),

Rule::Chain(4, TS('i'), TS('('), NS('W'), TS(')'))

),

Rule(NS('W'), GRB\_ERROR\_SERIES + 5, 4, // принимаемые параметры ф-ии

Rule::Chain(1, TS('i')),

Rule::Chain(1, TS('l')),

Rule::Chain(3, TS('i'), TS(','), NS('W')),

Rule::Chain(3, TS('l'), TS(','), NS('W'))

),

Rule(NS('M'), GRB\_ERROR\_SERIES + 1, 14, // знаки

Rule::Chain(2, TS('\*'), NS('E')),

Rule::Chain(2, TS('/'), NS('E')),

Rule::Chain(2, TS('+'), NS('E')),

Rule::Chain(2, TS('-'), NS('E')),

Rule::Chain(3, TS('+'), NS('E'), NS('M')),

Rule::Chain(3, TS('-'), NS('E'), NS('M')),

Rule::Chain(3, TS('\*'), NS('E'), NS('M')),

Rule::Chain(3, TS('/'), NS('E'), NS('M')),

Rule::Chain(3, TS('|'), NS('E'), NS('M')),

Rule::Chain(2, TS('|'), NS('E')),

Rule::Chain(2, TS('^'), NS('E')),

Rule::Chain(2, TS('&'), NS('E')),

Rule::Chain(3, TS('^'), NS('E'), NS('M')),

Rule::Chain(3, TS('&'), NS('E'), NS('M'))

),

Rule(NS('B'), GRB\_ERROR\_SERIES + 1, 1, // знаки для унарных операций

Rule::Chain(1, TS('~'))

)

);

# Рис.11 - Структура данных грамматики Грейбах

# Приложение Д

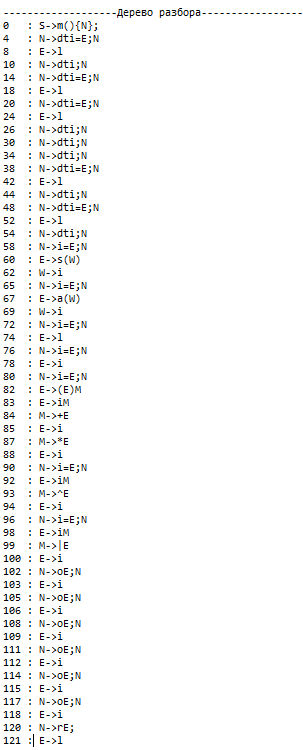


Рис. 12 - Дерево разбора

## Приложение Е

Таблица 1 – Результат генерации кода

|  |
| --- |
| .586  .model flat, stdcall  includelib kernel32.lib  includelib libucrt.lib  includelib D:\Trash\КП\new\LP\_Lab16\Debug\rand.lib  Rand PROTO : DWORD  Atoi PROTO : DWORD  ExitProcess PROTO : DWORD  GetStdHandle PROTO : DWORD  WriteConsoleA PROTO :DWORD, :DWORD, :DWORD, :DWORD, :DWORD  printConsole PROTO pstr: dword, \_size: dword  .stack 4096  .const  L0 dd 15  L1 dd 8  L2 dd 0116  L3 dd 0101  L4 dd 1  L5 db "e", 0  L6 db "a", 0  .data  x dd 15  y dd 8  d dd 116  z dd 101  r dd 0  p dd 0  f dd 0  hf dd 1  s dd 0 , 0  e dd "e" , 0  i dd 0  .code  printConsole PROC pstr: dword, \_size: dword  push eax  INVOKE GetStdHandle, -11  INVOKE WriteConsoleA, eax, pstr, \_size, 0, 0  pop eax  ret  printConsole ENDP  intToChar PROC number : sdword, pstr : dword  push eax  push ebx  push ecx  push edi  push esi  mov edi, pstr  mov esi, 0  mov eax, number  cdq  mov ebx, 10  idiv ebx  test eax, 80000000h  jz plus  neg eax  neg edx  mov cl, '-'  mov[edi], cl  inc edi  plus :  push dx  inc esi  test eax, eax  jz fin  cdq  idiv ebx  jmp plus  fin :  mov ecx, esi  write :  pop bx  add bl, '0'  mov[edi], bl  inc edi  loop write  pop esi  pop edi  pop ecx  pop ebx  pop eax  ret  intToChar ENDP  main PROC  mov edx, L0  mov x, edx  mov edx, L1  mov y, edx  mov edx, L2  mov d, edx  mov edx, L3  mov z, edx  mov edx, L4  mov hf, edx  mov edx, offset L5  mov e, edx  INVOKE Rand, x  mov r, eax  INVOKE Atoi, ADDR e  mov i, eax  mov edx, offset L6  mov s, edx  mov edx, i  mov x, edx  mov edx, L0  mov x, edx  mov ebx,y  add x, ebx  mov ebx, x  mov z, ebx  mov eax,y  mov ebx,x  mul ebx  mov z, eax  mov ebx, x  mov eax, y  xor ebx,eax  mov p, ebx  mov ebx, x  mov eax, i  or ebx, eax  mov f, ebx  INVOKE intToChar, z, ADDR z  INVOKE printConsole, ADDR z, 4  INVOKE intToChar, f, ADDR f  INVOKE printConsole, ADDR f, 4  INVOKE intToChar, p, ADDR p  INVOKE printConsole, ADDR p, 4  INVOKE intToChar, i, ADDR i  INVOKE printConsole, ADDR i, 4  INVOKE intToChar, r, ADDR r  INVOKE printConsole, ADDR r, 4  INVOKE printConsole, s, 2  mov eax, 0  ret  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