1. Спецификация языка программирования
   1. Характеристика языка программирования

Язык программирования LVS-2019 – строго типизированный, компилируемый язык. Не является объектно-ориентированным.

* 1. Алфавит языка

Алфавит языка LVS-2019 основан на кодировке Windows-1251, представленный на рисунке 1.1.

Исходный код LVS–2019 может содержать символы латинского алфавита малого и верхнего регистра ({a..z}, {A..Z}), цифры десятичной системы счисления от 0 до 9({0..9}), в строковых литералах разрешены символы латинского и русского алфавитов, а также знаки препинания.

* 1. Применяемые сепараторы

Символы-сепараторы – символы, используемые для разделения отдельных лексических единиц или функциональных элементов в исходном коде программы.

Символы–сепараторы представлены в таблице 1.1.

Таблица 1.1. Сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение |
| ;  пробел | Разделение инструкций |
| , | Разделение параметров функций |
| {  } | Программный блок |
| (  ) | Параметры и приоритетность операций(в выражениях)  Запись инструкций условных блоках (if()) |

* 1. Применяемые кодировки

Для написания исходного кода на языке программирования LVS-2019 используется кодировка Windows-1251, пример таблицы кодировок рис. 1.1

Рисунок 1.1. Алфавит входных символов

* 1. Типы данных

В языке LVA-2019 есть 2 типа данных: беззнаковый целый и символьные. Описание типов данных, предусмотренных в данном языке представлено в таблице 1.2.

Таблица 1.2. Типы данных языка программирования LVA-2019

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| 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

Функция стандартной библиотеки atoi() преобразует символ в число согласно его номеру в таблице Windows-1251. Другие преобразования типов не поддерживаются.

* 1. Идентификаторы

В имени идентификатора допускаются символы латинского алфавита нижнего и верхнего регистра. Также имя идентификатора может содержать цифры, но цифра не должна находиться на первой позиции. Максимальная длина идентификатора равна 5. Если его длина будет превышать максимальное значение, то имя идентификатора будет урезаться.

* 1. Литералы

Литерал – константное значение, которое непосредственно представляет некоторое значение. В языке существует 2 типа литералов: целого типа и символьные литералы. Краткое описание литералов представлено в таблице 1.3.

Таблица 1.3. Описание литералов

|  |  |
| --- | --- |
| Тип литерала | Описание |
| Литералы целого типа | Целочисленные литералы десятичного типа. Не имеют дробных частей или экспонент. |
| Символьные литералы | Символы, заключённые в “ “ (двойные кавычки), инициализируются как перемене типа char. |

* 1. Объявление данных

Переменные объявляются при помощи конструкции:

dec <тип данных> <идентификатор>;

Область видимости распространяется в пределах своего блока.

* 1. Инициализация данных

Для инициализации переменной каким-либо значением применяется конструкция:

<идентификатор>=<значение>;

Объявление по умолчанию см. таблица 1.2

* 1. Инструкции языка

Все возможные инструкции языка программирования LVA-2019 представлены в таблице 1.4.

Таблица 1.4. Инструкции языка программирования LVA-2019

|  |  |
| --- | --- |
| Инструкция | Синтаксис на языке программирования LVA-2019 |
| Главная функция | main(){  } |
| Объявление переменной | <тип данных> <идентификатор>; |
| Присваивание | <идентификатор>=<значение>; |
| Объявление функции | Используются только функции стандартной библиотеки  Объявление не требуется |
| Условный оператор | if(условие)  {  …  } |
| Возврат данных | return <идентификатор>|<литерал>; |
| Вывод данных | print <идентификатор>|<литерал>; |
| Функция стандартной библиотеки: atoi() | atoi(<идентификатор>); |
| Функция стандартной библиотеки: rand() | Rand(max); |

* 1. Операции языка

Язык программирования LVA-2019 может выполнять арифметические операции, представленные в таблице 1.5.

|  |  |
| --- | --- |
| Операция | Приоритетность операции |
| (  ) | 1 |
| \*  / | 2 |
| +  - | 3 |

Максимальный приоритет имеют операции с значением 1, минимальным 3 соответственно.

Побитовые операции над числом не могут применяться в выражениях:

|  |  |
| --- | --- |
| ~ | 1 |
| &  ^ | 2 |

Таблица 1.5.1

* 1. Выражения и их вычисления

Выражением называется совокупность переменных, знаков операций, имён функций, скобок, которая может быть вычислена в соответствии с синтаксисом языка программирования. Результатом вычисления выражения является величина определённого типа. Круглые скобки в выражении используются для изменения приоритета операций. Не допускается запись двух подряд идущих арифметических операций.

* 1. Конструкции языка

Ключевые программные конструкции языка программирования LVA-2019 представлены в таблице 1.6.

Таблица 1.6. Программные конструкции языка программирования LVA-2019

|  |  |
| --- | --- |
| Название | Синтаксис |
| Главная функция | main(){  } |

* 1. Область видимости идентификаторов

Область видимости языка программирования LVA-2019 работает сверху вниз внутри блока, где произведен вызов.

* 1. Семантические проверки

Семантические проверки невозможно проверить с помощью грамматик. На этапе семантического анализа код проверяется на наличие следующих ошибок:

* типы данных в выражении не совпадают
* недопустимый тип данных в выражении
* отсутствует точка входа
* обнаружено несколько точек входа
  1. Распределение оперативной памяти на этапе выполнения

Транслированный код использует две области памяти: сегмент констант(литералы) и сегмент данных(переменные и параметры функций).

* 1. Стандартная библиотека и её вызов

Функции стандартной библиотеки и их описание представлено в таблице 1.7.

Таблица 1.7. Состав стандартной библиотеки

|  |  |  |  |
| --- | --- | --- | --- |
| Имя функции | Возвращаемое значение | Параметры функции | Описание |
| Atoi(<идентификатор>) | int | Идентификатор типа char | Функция возвращает значение, согласно номеру символа в таблице Windows-1251 |
| Rand(min, max) | int | Max – максимальное значение | Функция возвращает случайное число в диапазоне от 0 до max |

* 1. Ввод и вывод данных

Вывод данных осуществляется при помощи ключевого слова print, после него следует та информация, которую необходимо вывести идентификатор/ “символ“

* 1. Точка входа

Точкой входа является функция main().

* 1. Препроцессор

В языке LVS-2019 препроцессоры не предусмотрены.

* 1. Соглашения о вызовах
  2. При генерации кода используется соглашение \_stdcall в котором все аргументы функцией передаются через стек, справа налево. Очистку стека производит вызываемая подпрограмма.
  3. Объектный код

Язык программирования LVA-2019 транслируется в язык Assembler.

* 1. Классификация сообщений транслятора

[Системные]ХХХ XXX – код ошибки. Сообщение, генерируемое при критической ошибке

[Лексические]XXX ХХХ - код ошибки. Сообщение, генерируемое на этапе лексического анализа.

[Синтаксические]XXX ХХХ - код ошибки. Сообщение, генерируемое на этапе синтаксического анализа.

[Семантические]XXX ХХХ - код ошибки. Сообщение, генерируемое на этапе семантического анализа.

* 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 поступает таблица лексем, полученная на этапе лексического анализа. Если программа построена синтаксически правильно, то осуществляется переход к этапу трансляции стоящему далее, при ином раскладе работа транслятора останавливается.

Наборы функций, проверяющие правила на разных этапах работы транслятора 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, таблица литералов, 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.

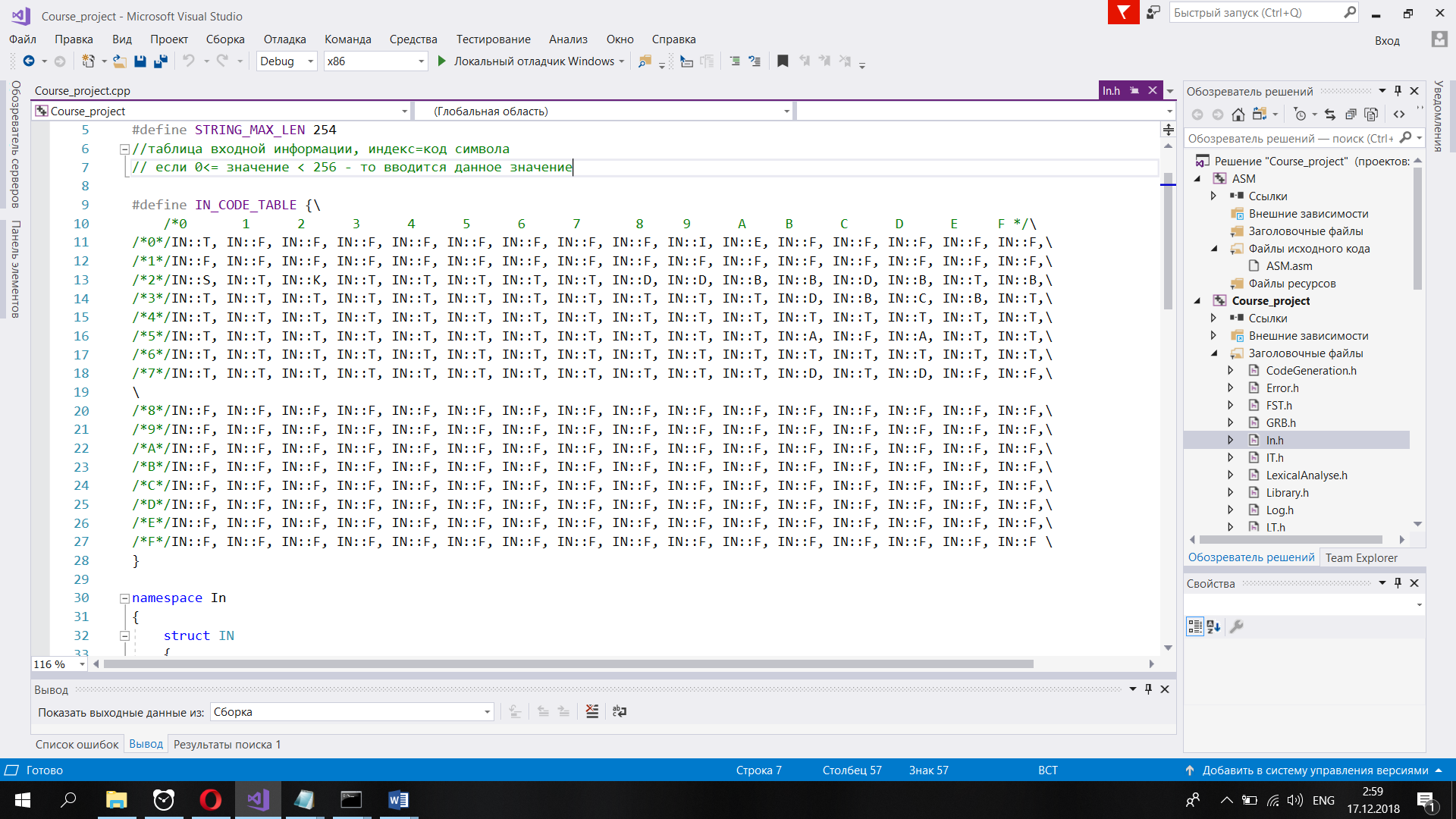


Рисунок 3.2 - Таблица допустимости входных символов

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 «QUOTE», 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 «SEP», то заносим слово в массив токенов.

1.3) Если «NS», то инкрементируем переменную, то есть увеличиваем на единицу, отвечающую за подсчет строк.

1.4) 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 |
| num | t |
| char | t |
| out(print) | 0 |
| return | r |
| main | m |
| dec | d |
| + | + |
| - | - |
| \* | \* |
| / | / |
| = | = |
| ( | ( |
| ) | ) |
| { | { |
| } | } |
| , | , |
| ; | ; |
| идентификатор | i |
| числовой литерал | l |
| 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 Подсчет количества ошибок не ведется.

## 3.7 Структура и перечень сообщений лексического анализатора

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

Таблица 3.2 - Сообщения лексического анализатора

|  |  |
| --- | --- |
| Код | Сообщение |
| 5 | Неопознанный символ |
| 103 | Попытка переопределения |
| 105 | Отсутствует предварительное определение |
| 111 | Недопустимый символ в исходном файле (-in) |
| 117 | Превышен размер таблицы лексем |
| 124 | Не удалось разобрать слово |

**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

|  |  |
| --- | --- |
| Нетереминалы | Описание |
| S | Правила, описывающие общую структуру программы |
| N | Порождает правила, описывающие инструкции языка |
| E | Порождает правила, описывающие выражения |
| M | Порождает правила, описывающие арифметические действия(бинарные) |
| F | Порождает правила, описывающие формальные параметры функции |
| W | Порождает правила, описывающие фактические параметры функции |
| B | Порождает правила, описывающие арифметические действия (унарные) |

## 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 перехода, 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 “[Синтаксическая]:”. Перечень сообщений, генерируемых на этапе синтаксического анализа, представлен в таблице 4.3.

Таблица 4.3 - Перечень сообщений

|  |  |
| --- | --- |
| Код | Сообщение |
| 8 | Ошибка в синтакическом анализе. Неизвестное правило разбора!Сверьтесь с разрешёнными комбинациями цепочек |
| 600 | Неверная структура программы |
| 602 | Ошибка в выражении |
| 603 | Ошибка в параметрах вызываемой функции |
| 604 | Ошибка в конструкции 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, синтаксического анализатора 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.

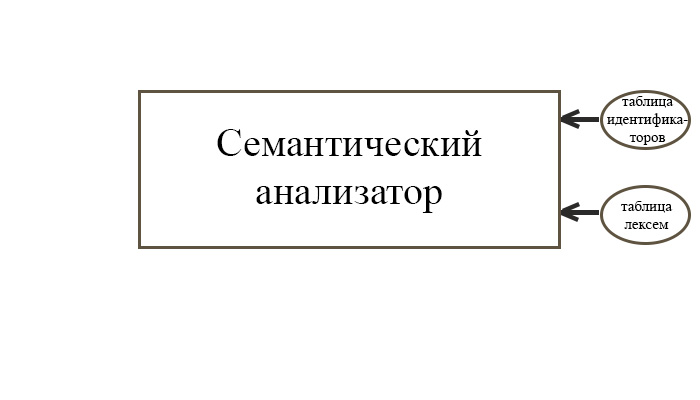


Рисунок 5.1 - Структура семантического анализатора

* 1. **Функции семантического анализатора**

bool SemanticAnalyze, запускается перед синтаксическим анализатором

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

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

Таблица 5.2 - Перечень сообщений

|  |  |
| --- | --- |
| Код | Сообщение |
| 3 | Несоответствие типов |
| 4 | Несоответствие присваиваемого типа типу переменной |
| 6 | Число выходит за границы допустимого |
| 7 | Не хватает закрывающей кавычки |
| 101 | Отсутствует главная функция |
| 109 | Обнаруженно невозможное деление на ноль числа или попытка деления строки на ноль |
| 114 | Несоответствие типов |
| 118 | Обнаружена вторая главная функция |
| 123 | Отсутствие главной функции |
| 605 | Использование встроенной функции без подключения библиотеки |

* 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 MAA-2018 допускаются выражения с использованием числовых идентификаторов и литералов. Также предусмотрены следующие арифметические операции:

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

Также предусмотрены следующие побитовые операции:

* и: &
* или: ||
* не: ~

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.

**6.4 Контрольный пример**

Контрольный пример разбора выражения содержится в таблице 6.2.

Таблица 6.2 - Разбор выражения

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

# Глава 7. Генерация кода

## 7.1 Структура генератора кода

Заключительным этапом трансляции языка MAA-2018 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 MAA-2018 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 MAA-2018 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка MAA-2018 и языка ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке MAA-2018 | Тип идентификатора на языке ассемблера | Пояснение |
| int | DWORD | Хранит целочисленный тип данных. |
| char | BYTE | Хранит указатель на начало строки. Строка должна завешаться нулевым символом. |

## 7.3 Статическая библиотека

В языке MAA-2018 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++(extern “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 |
| Long Rand( long k) | Возращает случайное число 0< I <k |
| Long atoi(char k) | Преобразует символ в число согласно табл. Windows – 1251 |

## 7.4 Особенности алгоритма генерации кода

В языке MAA-2018 генерация кода строится на основе таблиц лексем и идентификаторов. Общая схема работы генератора кода представлена на рисунке 7.2

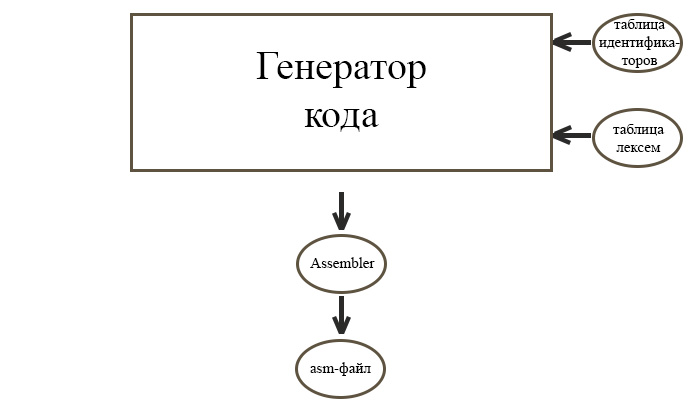


Рисунок 7.2 - Структура генератора кода

## 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

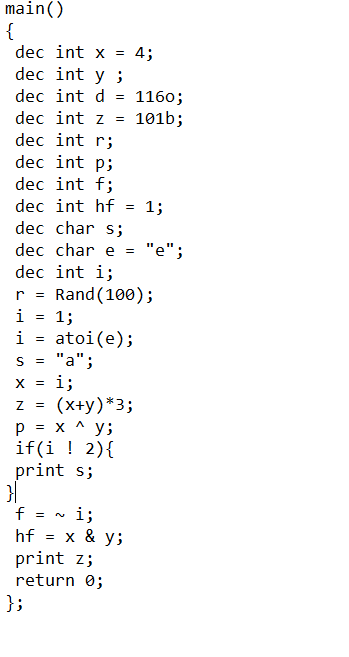
|  |  |
| --- | --- |
| Исходный код | Генерируемая ошибка |
| main()  {  dec char a = "a;  } | Ошибка 113: [Синтаксическая]Отсутствуют закрывающие кавычки. Строка: 3 |
| main  {  dec char abcdabcdabcd = "s";  } | Ошибка  [Лексическая]Превышена максимальная длина идентификатора. Усечение до 8. Строка: 3. |
|  |  |
|  |  |

# 

# Заключение

* Реализованы 7 оператора для вычисления выражений;
* Реализованы два типа данных;
* реализована преобразование
* поддерживается оператор вывода;
* Реализованы 2 функций для двух типов данных;
* присутствует подключаемая стандартная библиотека;
* обрабатывается 30 возможных исключительных ситуаций;
* поддерживается оператор вывода.

# Приложение А

 Рис. 8 - Исходный код

# Приложение Б

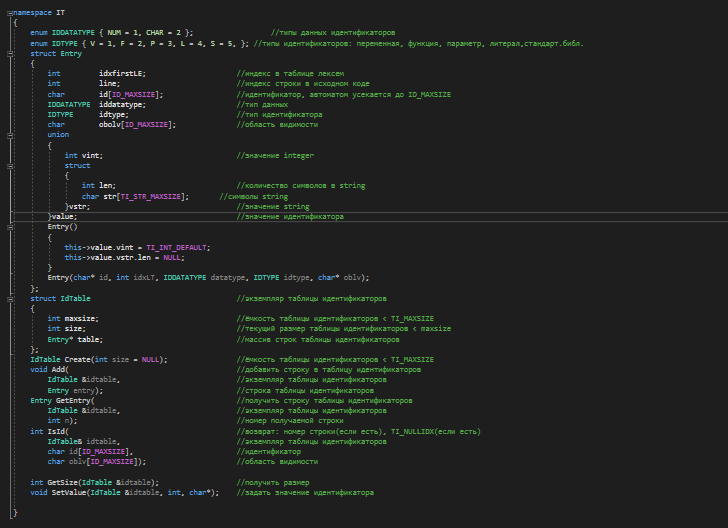
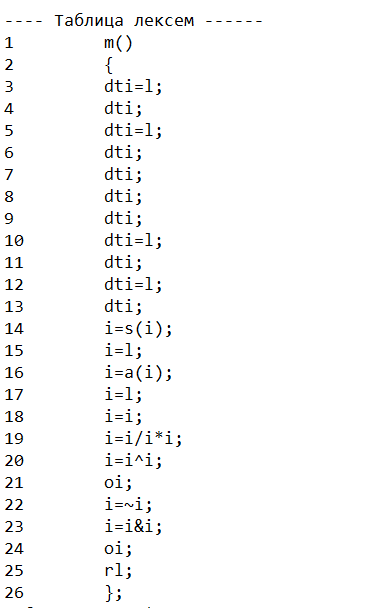
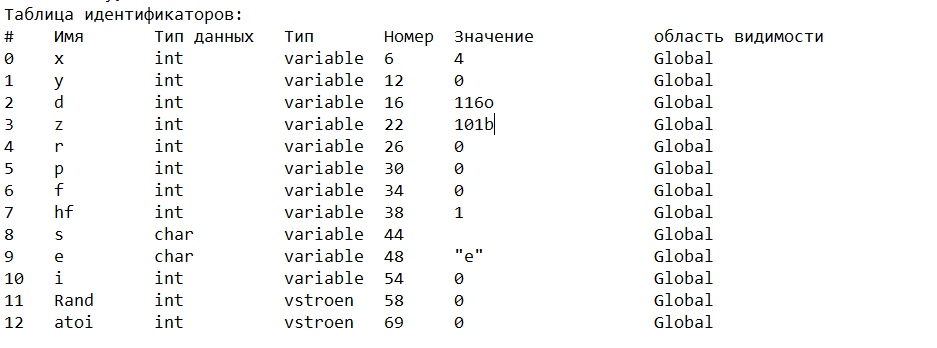


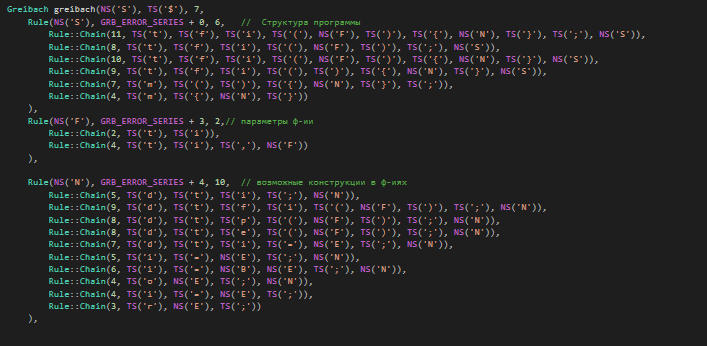
Рис. 9 - Структуры данных, используемые на фазе лексического анализа

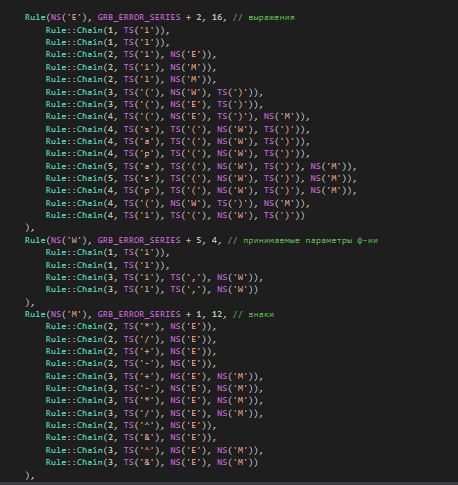
# Приложение В





# Приложение Г







# Приложение Д

