**Содержание**

[**Введение** 5](#_Toc27617979)

[**1** **Спецификация языка программирования** 6](#_Toc27617980)

[**1.1** **Характеристика языка программирования** 6](#_Toc27617981)

[**1.2** **Определение алфавита языка программирования** 6](#_Toc27617982)

[**1.3** **Применяемые сепараторы** 7](#_Toc27617983)

[**1.4** **Применяемые кодировки** 7](#_Toc27617984)

[**1.5** **Типы данных** 7](#_Toc27617985)

[**1.6** **Преобразование типов данных** 8](#_Toc27617986)

[**1.7** **Идентификаторы** 8](#_Toc27617987)

[**1.8** **Литералы** 8](#_Toc27617988)

[**1.9** **Объявление данных** 9](#_Toc27617989)

[**1.10** **Инициализация данных** 10](#_Toc27617990)

[**1.11** **Инструкции языка** 10](#_Toc27617991)

[**1.12** **Операции языка** 10](#_Toc27617992)

[**1.13** **Выражения и их вычисления** 11](#_Toc27617993)

[**1.14** **Конструкции языка** 12](#_Toc27617994)

[**1.15** **Область видимости идентификатора** 12](#_Toc27617995)

[**1.16** **Семантические проверки** 13](#_Toc27617996)

[**1.17** **Распределение оперативной памяти на этапе выполнения** 13](#_Toc27617997)

[**1.18** **Стандартная библиотека и её состав** 13](#_Toc27617998)

[**1.19** **Ввод и вывод данных** 14](#_Toc27617999)

[**1.20** **Точка входа** 14](#_Toc27618000)

[**1.21** **Препроцессор** 14](#_Toc27618001)

[**1.22** **Соглашение о вызовах** 14](#_Toc27618002)

[**1.23** **Объектный код** 14](#_Toc27618003)

[**1.24** **Классификация сообщений транслятора** 15](#_Toc27618004)

[**1.25** **Контрольный пример** 15](#_Toc27618005)

[**2** **Структура транслятора** 16](#_Toc27618006)

[**2.1** **Компоненты транслятора их назначение** 16](#_Toc27618007)

[**2.2** **Перечень входных параметров транслятора** 17](#_Toc27618008)

[**2.3** **Протоколы, формируемые транслятором** 17](#_Toc27618009)

[**3** **Разработка лексического анализатора** 19](#_Toc27618010)

[**3.1** **Структура лексического анализатора** 19](#_Toc27618011)

[**3.2** **Контроль входных символов** 19](#_Toc27618012)

[**3.3** **Удаление избыточных символов** 20](#_Toc27618013)

[**3.4** **Перечень ключевых слов** 20](#_Toc27618014)

[**3.5** **Основные структуры данных** 22](#_Toc27618015)

[**3.6** **Структура и перечень сообщений лексического анализатора** 24](#_Toc27618016)

[**3.7** **Принцип обработки ошибок** 25](#_Toc27618017)

[**3.8** **Параметры лексического анализатора** 25](#_Toc27618018)

[**3.9** **Алгоритм лексического анализа** 25](#_Toc27618019)

[**3.10** **Контрольный пример** 26](#_Toc27618020)

[**4** **Разработка синтаксического анализатора** 27](#_Toc27618021)

[**4.1** **Структура синтаксического анализатора** 27](#_Toc27618022)

[**4.2** **Грамматика, описывающая синтаксис языка** 27](#_Toc27618023)

[**4.3** **Построение конечного магазинного автомата** 30](#_Toc27618024)

[**4.4** **Основные структуры данных** 31](#_Toc27618025)

[**4.5** **Описание алгоритма синтаксического разбора** 31](#_Toc27618026)

[**4.6** **Структура и перечень сообщений синтаксического анализатора** 32](#_Toc27618027)

[**4.7** **Параметры синтаксического анализатора и режимы его работы** 32](#_Toc27618028)

[**4.8** **Принцип обработки ошибок** 32](#_Toc27618029)

[**4.9** **Контрольный пример** 32](#_Toc27618030)

[**5** **Разработка семантического анализатора** 33](#_Toc27618031)

[**5.1** **Структура семантического анализатора** 33](#_Toc27618032)

[**5.2** **Функции семантического анализатора** 33](#_Toc27618033)

[**5.3** **Структура и перечень сообщений семантического анализатора** 34](#_Toc27618034)

[**5.4** **Принцип обработки ошибок** 34](#_Toc27618035)

[**5.5** **Контрольный пример** 35](#_Toc27618036)

[**6** **Вычисление выражений** 36](#_Toc27618037)

[**6.1** **Выражение, допускаемые языком** 36](#_Toc27618038)

[**6.2** **Польская запись и принцип её построения** 36](#_Toc27618039)

[**6.3** **Программная реализация обработки выражений** 37](#_Toc27618040)

[**6.4** **Контрольный пример** 37](#_Toc27618041)

[**7** **Генерация кода** 38](#_Toc27618042)

[**7.1** **Структура генератора кода** 38](#_Toc27618043)

[**7.2** **Представление типов данных в оперативной памяти** 38](#_Toc27618044)

[**7.3** **Статическая библиотека** 39](#_Toc27618045)

[**7.4** **Особенности алгоритма генерации кода** 39](#_Toc27618046)

[**7.5** **Входные параметры генератора** 40](#_Toc27618047)

[**7.6** **Контрольный пример** 40](#_Toc27618048)

[**8** **Тестирование транслятора** 41](#_Toc27618049)

[**8.1** **Общие положения** 41](#_Toc27618050)

[**8.2** **Результаты тестирования** 41](#_Toc27618051)

[**Заключение** 43](#_Toc27618052)

[**Список использованных источников** 44](#_Toc27618053)

[**Приложение А** 45](#_Toc27618054)

[**Приложение Б** 46](#_Toc27618055)

[**Приложение В** 49](#_Toc27618056)

[**Приложение Г** 50](#_Toc27618057)

[**Приложение Д** 51](#_Toc27618058)

# **Введение**

Целью курсового проекта является написание спецификации и разработка транслятора для собственного языка программирования.

Название языка ­­– PMN-2019. Трансляция будет производиться в язык ассемблера.

Этапы выполнения курсового проекта:

1. Написание спецификации языка программирования;
2. Разработка лексического анализатора;
3. Разработка синтаксического анализатора;
4. Разработка семантического анализатора;
5. Преобразование арифметических выражений;
6. Генерация кода;
7. Тестирование транслятора.

Подробная информация о каждом этапе выполнения курсового проекта приведена в соответствующих разделах пояснительной записки:

В первом разделе приведена спецификация языка.

Во втором разделе описана структура транслятора.

В третьем разделе описаны принцип работы и этапы разработки лексического анализатора, определены разрешенные символы и ключевые слова языка программирования.

В четвертом разделе описан принцип работы синтаксического анализатора, формальная грамматика определена и приведена в нормальную форму Грейбах для выполнения синтаксического разбора.

В пятом разделе описаны принцип работы и основные функции семантического анализатора.

В шестом разделе описаны выражения допускаемые языком, форма, принципы построения и вычисления выражений.

В седьмом разделе описан процесс генерации кода.

В восьмом разделе приведены примеры тестирования транслятора.

1. **Спецификация языка программирования**
   1. **Характеристика языка программирования**

Язык PMN-2019 является универсальным компилируемым строго типизированным процедурным высокоуровневым языком программирования, поддерживающим парадигму функционального программирования.

* 1. **Определение алфавита языка программирования**

Алфавит языка PMN-2019 основан на кодировке Windows-1251, изображенной на рисунке 1.1.



Рисунок 1.1 Алфавит входных символов языка PMN-2019

Символы, используемые на этапе выполнения, на рисунке помечены как разрешенные.

* 1. **Применяемые сепараторы**

Сепараторы, применяемые в языке PMN-2019, приведены в таблице 1.1.

Таблица 1.1 Применяемые сепараторы

|  |  |
| --- | --- |
| Разделители | Назначение |
| ‘пробел’, ‘табуляция’,  ‘переход на новую строку’ | Разделяют входные лексемы |
| +, -, \*, /, % | Арифметические операторы. Используются в арифметических операциях |
| = | Оператор присваивания. Используется для присваивания значения переменной |
| <, >, !=, == | Условные операторы. Используются для сравнения переменных и литералов |
| ( ) | Блок параметров функции, так же указывает приоритет в арифметических операциях |
| , | Разделяет параметры функции |
| { } | Ограничивают программные конструкции |
| ; | Признак конца инструкции языка |

* 1. **Применяемые кодировки**

Для написания кода на языке программирования PMN-2019 используется кодировка Windows-1251.

* 1. **Типы данных**

В языке PMN-2019 поддерживается 3 типа данных: целочисленный, строковый и булевый. Подробная описание типов данных приведено в таблице 1.2.

Таблица 1.2 Типы данных языка PMN-2019

| Тип данных | Характеристика |
| --- | --- |
| Целочисленный  (dig) | В памяти занимает 1 байт.  Максимальное значение: 127.  Минимальное значение: -128.  Принцип размещения в памяти:  Последний бит числа отведен под знак, оставшиеся 7 бит предназначены для хранения значения числа.  Значение по умолчанию: 0.  Применяемые операции: в арифметических выражениях и условных конструкциях к целочисленным переменным и литералам применимы все арифметические и условные операции соответственно, поддерживаемые языком PMN-2019. |

Продолжение таблицы 1.2

|  |  |
| --- | --- |
| Тип данных | Характеристика |
| Строковый  (str) | В памяти занимает n + 1 байт, где n – количество символов в строке.  Максимальное количество символов: 255.  Принцип размещения в памяти:  Каждый символ строки занимает 1 байт. В конце строки располагается NULL символ (признак конца строки).  Применяемые операции: к строковым переменным и литералам операции не применяются. |
| Булевый  (bool) | В памяти занимает 1 байт.  Может принимать одно из двух значений: true или false.  Принцип размещения в памяти:  В зависимости от значения 1 бит числа установлен true: 1, false: 0.  Применяемые операции: к булевым переменным и литералам применимы все условные операции, поддерживаемые языком PMN-2019. |

* 1. **Преобразование типов данных**

Преобразования типов данных в языке программирования PMN-2019 не допустимо.

* 1. **Идентификаторы**

Идентификатор языка PMN-2019 – это имя, используемое для переменных, функций, параметров. Идентификаторы могут состоять как из одного, так и из нескольких символов. Первым символом должна быть маленькая буква латинского алфавита, а за ним могут стоять маленькие буквы латинского алфавита или цифры. Так же идентификаторы не могут совпадать с ключевыми словами языка PMN-2019.

Пример корректных идентификаторов: stroka1, fdsgf141 и т.п.

Пример некорректных идентификаторов: 14strok, Stroka1, \_stroka1 и т.п.

* 1. **Литералы**

Литерал языка PMN-2019 – это запись в исходном коде программы, представляющая собой фиксированное значение. Описание литералов приведено в таблице 1.3.

Таблица 1.3 Литералы языка PMN-2019

|  |  |
| --- | --- |
| Тип литерала | Характеристика |
| Целочисленный | Десятичный:  Последовательность десятичных цифр 0..9 с предшествующим знаком минус или без него.  Двоичный:  Последовательность двоичных цифр 0 и 1 с предшествующим знаком минус или без него, в конце которой стоит символ ‘B’ (признак двоичного целочисленного литерала).  Восьмеричный:  Последовательность восьмеричных цифр 0..7 с предшествующим знаком минус или без него, в конце которой стоит символ ‘O’ (признак восьмеричного целочисленного литерала).  Шестнадцатеричный:  Последовательность шестнадцатеричных чисел 0..F с предшествующим знаком минус или без него, в конце которой стоит символ ‘H’ (признак шестнадцатеричного целочисленного литерала).  Допустимый диапазон значений:  От -128 до 127 в десятичной системе исчисления. |
| Строковый | Набор, состоящий из символов русского и латинского алфавитов и десятичных цифр, заключенный в двойные кавычки.  Допустимый диапазон значений:  От 0 до 255 символов. |
| Булевый | true или false, где true: логическая единица, false: логический ноль.  Допустимый диапазон значений:  true или false. |

Пример корректных литералов: 23, -11, 4FH, -1001B, 34O, false, “sroka” и т.п.

Примеры некорректных литералов: -4А, ‘sroka’, 1010, trui и т.п.

* 1. **Объявление данных**

Для объявления переменной используется ключевое слов new, после которого указывается тип переменной и имя идентификатора. Так же при объявлении допускается инициализация переменной.

new <тип> <имя идентификатора>;

new <тип> <имя идентификатора> = <литерал>;

Переменную можно объявить в блоке main, в блоке функции, в условном блоке if..else или в блоке while. Область видимости переменной будет соответствовать тому блоку, в котором она находиться.

* 1. **Инициализация данных**

В языке PMN-2019 присутствует 2 вида инициализации:

1. Инициализация в месте объявления

new <тип> <имя идентификатора> = <литерал>;

1. Инициализация после объявления

<имя идентификатора> = <литерал>;

Так же в языке присутствует инициализация по умолчанию. Переменные типа dig по умолчанию инициализируются нулем. Переменные типа str по умолчанию инициализируются пустой строкой. Переменные типа bool по умолчанию инициализируются значение false: логический ноль.

* 1. **Инструкции языка**

Инструкции языка PMN-2019 приведены в таблице 1.4.

Таблица 1.4 Инструкции языка PMN-2019

|  |  |
| --- | --- |
| Инструкция | Форма записи |
| Объявление переменной | **type** <тип данных> <идентификатор>. |
| Объявление переменной с явной инициализацией | **type** <тип данных> <идентификатор> = <значение>|<выражение>;  Значение – литерал, идентификатор, вызов функции соответствующего типа данных |
| Объявление функции | <тип данных> **function** <идентификатор> (<тип данных> <идентификатор>, …)  {  / программный блок /  **return** <идентификатор/литерал>.  } |
| Вызов функции | <идентификатор> (<идентификатор>, ...) |
| Присвоение значения | <идентификатор> = <значение>; |
| Вывод данных | **out** <идентификатор/литерал>; |
| Вывод данных с переходом на новую строку | **outl** <идентификатор/литерал>; |
| Возвращаемое значение | **return** <литерал/идентификатор>. |

* 1. **Операции языка**

Операции языка PMN-2019 приведены в таблице 1.5.

Наибольшая приоритетность у операций умножения, деления и деления с остатком, затем идут операции сложения и вычитания. Помещение операции в скобки задают ей самый высокий приоритет.

Таблица 1.5 Операции языка PMN-2019

|  |  |
| --- | --- |
| Тип оператора | Оператор |
| Арифметические операции языка | + – сложение  - – вычитание  \* – умножение  / – деление  % – остаток от деления |
| Логические | > – больше  < – меньше  == – проверка на равенство  != – проверка на неравенство |

* 1. **Выражения и их вычисления**

Выражение языка программирования PMN-2019 – это совокупность переменных, литералов, вызовов функций, знаков операций, скобок, которая может быть вычислена в соответствии с синтаксисом языка.

Правила составления выражений:

1. Выражения записываются в одну строку;
2. В выражении могут присутствовать только операнды одинакового типа;
3. В выражении могут использоваться функции. Как стандартные, так и пользовательские;
4. В выражении не могут идти подряд два оператора;
5. Допускается использование круглых скобок для смены приоритета операций.

В арифметических выражениях допускаются только операнды целочисленного типа. В выражениях сравнения допускаются операнды булевого и целочисленного типов.

Перед генерацией кода выражения приводятся к ПОЛИЗ для более удобного вычисления на языке ассемблера.

* 1. **Конструкции языка**

Конструкции языка PMN-2019 приведены в таблице 1.6.

Таблица 1.6 Конструкции языка PMN-2019

|  |  |
| --- | --- |
| Конструкция | Описание |
| Главная функция | main  {  …  return <имя переменной/литерал>;  }; |
| Пользовательская функция | <тип возвращаемого значения> function(<тип параметра> <имя параметра>, …)  {  …  return <имя переменной/литерал>;  };  Максимальное количество параметров: 8. |
| Условная конструкция | if(<имя переменной/литерал><условный оператор><имя переменной/литерал>)  {  …  }  else  {  ...  }  Блок else необязателен. |
| Конструкция цикла | while(<имя переменной/литерал><условный оператор><имя переменной/литерал>)  {  …  } |

* 1. **Область видимости идентификатора**

Каждой конструкции языка PMN-2019 соответствует своя область видимости. Причем функции имеют глобальную область видимости. Однако внутри главной и пользовательских функций действует принцип сокрытия переменных. А именно: переменные, определенные внутри условной конструкции или конструкции цикла могут скрывать внешние переменные с тем же именем.

Объявление идентификаторов вне функции невозможно, так как глобальные переменные языком не поддерживаются.

* 1. **Семантические проверки**

Семантическим анализатором языка PMN-2019 предусмотрены следующие проверки:

1. Наличие блока main, точки входа в программу;
2. Единственная точка входа в программу;
3. Использование идентификаторов до их объявления;
4. Переопределение идентификаторов;
5. Соответствие параметров, передаваемых в функцию, с параметрами в объявлении функции;
6. Соответствие типа возвращаемого значения с типом функции;
7. Соответствие типов в выражениях;
8. Превышение размера целочисленных и строковых литералов;
9. Превышение длины лексемы;
10. Перегрузка оператора для работы со строками.
    1. **Распределение оперативной памяти на этапе выполнения**

Для запоминания промежуточных результатов в вычислении выражения используется стек. В сегмент констант записываются все литералы языка. В сегмент данных записываются все имена переменных.

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

В языке PMN-2019 предусмотрена стандартная библиотека, которая включает в себя набор стандартных функций, а так же функций вывода в консоль. Функции, входящие в состав стандартной библиотеки приведены в таблице 1.7.

Таблица 1.7 Функции стандартной библиотеки языка PMN-2019

|  |  |
| --- | --- |
| Прототип функции | Описание |
| power(dig x, dig y); | Возводит число x в степень y и возвращает результат. |
| abser(dig x); | Берет абсолютное значение числа x и возвращает результат. |
| nout(dig x); | Выводит целочисленный идентификатор или литерал на консоль. |
| sout(str x); | Выводит строковый идентификатор или литерал на консоль. |
| noutl(dig x); | Выводит целочисленный идентификатор или литерал на консоль, переводя каретку на новую строку. |
| soutl(str x); | Выводит строковый идентификатор или литерал на консоль, переводя каретку на новую строку. |

Стандартная библиотека написана на языке С++, подключается к транслированному коду на этапе генерации кода. Вызовы стандартных функций доступны там же, где и вызов пользовательских функций.

* 1. **Ввод и вывод данных**

Вывод данных на консоль осуществляется за счет операторов **out** и **outl**. Использование данных операторов допускается только с идентификаторами или литералами.

Функции, управляющие выводом данных на консоль, реализованы на языке C++. На этапе генерации кода операторы вывода языка PMN-2019 заменяются на встроенные функции, находящиеся в стандартной библиотеке.

* 1. **Точка входа**

В языке PMN-2019 точкой входа является блок главной функции **main.** С первой строчки которого начинается выполнение инструкций программы.

* 1. **Препроцессор**

Препроцессор в языке программирования PMN-2019 не предусмотрен.

* 1. **Соглашение о вызовах**

В языке PMN-2019 вызов функций происходит по соглашению о вызовах stdcall. Данное соглашение имеет следующие особенности:

1. Все параметры функции передаются через стек;
2. Освобождением памяти занимается вызываемый код;
3. Параметры в стек заносятся справа налево.
   1. **Объектный код**

Язык программирования PMN-2019 транслируется в язык ассемблера.

* 1. **Классификация сообщений транслятора**

Описание и классификация сообщений транслятора об ошибках приведено в таблице 1.8.

Таблица 1.8 Описание ошибок транслятора языка PMN-2019

|  |  |
| --- | --- |
| Интервал кодов | Описание |
| 0-9 | Системные ошибки. |
| 10-19 | Ошибки параметров. |
| 20-29 | Ошибки файлов. |
| 110-129 | Ошибки лексического анализатора. |
| 130-149 | Ошибки семантического анализатора. |
| 600-609 | Ошибки синтаксического анализатора. |

* 1. **Контрольный пример**

Контрольный пример языка PMN-2019 представлен в приложении А.

1. **Структура транслятора**
   1. **Компоненты транслятора их назначение**

Транслятор – это программа преобразующая исходный код на одном языке программирования в исходный код на другом языке.

Схема, поясняющая принцип работы транслятора, изображена на рисунке 2.1.

Трансляция исходного кода в язык ассемблера разделена на 4 этапа: Лексический анализ, синтаксический анализ, семантический анализ и генерация кода.

Этапы выполняются последовательно. У каждого этапа есть входные и выходные данные, которые тот или иной этап получает от предыдущей и передает следующей части трансляции.

Первой частью трансляции является лексический анализ. На вход лексического анализатора подается исходный код программы. В свою очередь лексический анализатор производит деление исходного кода программы на слова (токены), которые затем идентифицируются и заменяются на лексемы (внутреннее представление). На выходе лексического анализатора мы имеем две таблицы: таблицу лексем и таблицу идентификаторов.

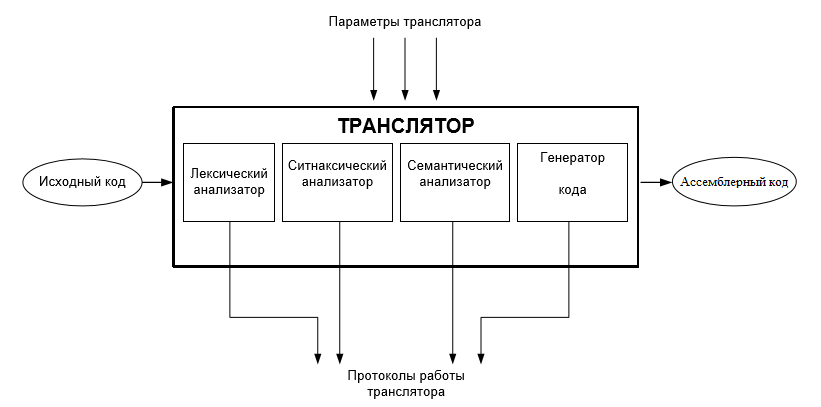


Рисунок 2.1 Структура транслятора языка PMN-2019

Синтаксический анализ является второй частью работы транслятора. Синтаксический анализатор выполняет синтаксический анализ. Входом для синтаксического анализатора является таблица лексем и таблица идентификаторов. Выходом – дерево разбора.

Затем выполняется семантический анализ. Задача семантического анализатора: проверка соблюдения в исходной программе семантических правил входного языка. Входом для семантического анализатора является таблица идентификаторов и дерево разбора.

Последним этапом трансляции является генерация кода. Вход для генератора когда: таблица лексем и таблица идентификаторов. Выход – кода на языке ассемблера.

* 1. **Перечень входных параметров транслятора**

Входные параметры транслятора языка программирования PMN-2019 приведены в таблице 2.1. Входные параметры необходимы для формирования файлов с результатами работы транслятора.

Таблица 2.1 Входные параметры транслятора языка PMN-2019

|  |  |  |
| --- | --- | --- |
| Ключ и входной параметр | Описание | Значение по умолчанию |
| -in:<путь к файлу> | Текстовый файл с исходным кодом на языке PMN-2019. | Отсутствует |
| -out:<путь к файлу> | Выходной файл, содержащий исходный код на языке ассемблера. | <имя файла in>.out |
| -log:<путь к файлу> | Файл с протоколом работы транслятора. | <имя файла in>.log |
| -tkn:<путь к файлу> | Файл, содержащий исходный текст программы во внутреннем представлении. | <имя файла in>.tkn |
| -lex:<путь к файлу> | Файл с таблицей лексем. | <имя файла in>.lex |
| -id:<путь к файлу> | Файл с таблицей идентификаторов. | <имя файла in>.id |
| -sin:<путь к файлу> | Файл с деревом разбора синтаксического анализатора. | <имя файла in>.sin |

* 1. **Протоколы, формируемые транслятором**

Протоколы, формируемые транслятором языка PMN-2019 приведены в таблице 2.2.

Таблица 2.2 Протоколы, формируемые транслятором языка PMN-2019

|  |  |
| --- | --- |
| Протокол | Описание |
| Файл, заданный параметром “-log:” | Содержит общую информацию о ходе выполнения трансляции: перечисление входных параметров, количество символов и строк, успех или ошибку по каждому этапу трансляции. В случае возникновения ошибки, в файл будет выведен номер ошибки. |
| Файл, заданный параметром “-tkn:” | Содержит исходный текст программы во внутреннем представлении языка. |
| Файл, заданный параметром “-lex:” | Содержит таблицу лексем, итог работы лексического анализа. |
| Файл, заданный параметром “-id:” | Содержит таблицу идентификаторов, итог работы лексического анализа. |
| Файл, заданный параметром “-sin:” | Содержит дерево разбора, итог работы синтаксического анализатора. |

1. **Разработка лексического анализатора**
   1. **Структура лексического анализатора**

Лексический анализатор – это программа, преобразующая исходный текст программы, заменяя лексические единицы их внутренним представлением – лексемами, для создания промежуточного представления исходной программы. Структурная схема лексического анализатора изображена на рисунке 3.1.

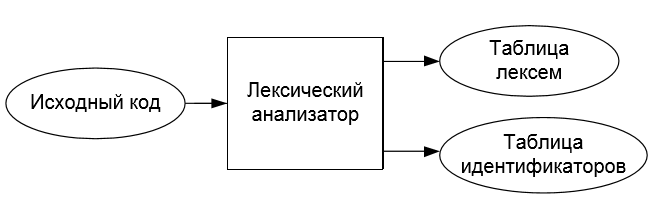


Рисунок 3.1 Структурная схема лексического анализатора

Лексический анализ в языке PMN-2019 состоит из двух частей:

1. Разбиение исходного кода программы на слова (токены).
2. Идентификация слов (токенов) и последующая их замена на лексемы (внутренне представление). Заполнение таблиц лексем и идентификаторов.

Входные данные: исходный код.

Результат работы: Таблица лексем и таблица идентификаторов.

* 1. **Контроль входных символов**

Для проверки входных символов на допустимость была написана таблица входных символов, которая дублируют таблицу кодировки Windows-1251. Разрешенные символы в таблице помечены символом T, запрещенные – F. Таблица контрольных символов языка PMN-2019 изображена на рисунке 3.2.

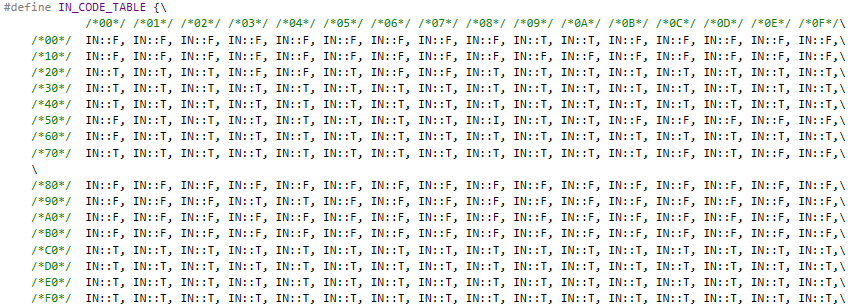


Рисунок 3.2 Таблица контрольных символов языка PMN-2019

* 1. **Удаление избыточных символов**

Избыточный символ – это символ, отсутствие которого никоем образом не влияет на исходный текст программы. В языке PMN-2019 символ пробела и табуляции являются избыточными символы. Их удаление предусмотрено на этапе разбиения исходного кода программы на слова (токены).

Алгоритм удаления избыточных символов

Пока есть символы для чтения:

* Читаем очередной символ;
* Если символ является пробелом или табуляцией:

1. Если идёт запись слова, пробел или табуляция являются символом сепаратором, сигнализирующем о начале или конце записи слова (токена);
2. Иначе символ пробела или табуляции пропускаются.
   1. **Перечень ключевых слов**

Все ключевые слова языка PMN-2019, сепараторы, символы операций, соответствующие им лексемы и регулярные выражения приведены в таблице 3.1.

Таблица 3.1 Все ключевые слова, сепараторы и т.д. языка PMN-2019

|  |  |  |
| --- | --- | --- |
| Слово (токен) | Лексема | Описание |
| dig | d | Целочисленный тип. |
| str | s | Строковый тип. |
| bool | b | Булевый тип. |
| идентификатор | i | Идентификатор любого типа языка. |
| литерал | l | Литерал любого типа языка. |
| while | w | Начало конструкции цикла. |
| if | q | Условный оператор и истинная ветвь условного оператора. |

Продолжение таблицы 3.1

|  |  |  |
| --- | --- | --- |
| Слово (токен) | Лексема | Описание |
| else | 2 | Ложная ветвь условного оператора. |
| function | f | Начало объявления функции. |
| new | n | Объявление переменной. |
| return | r | Выходи из функции и возврат значения. |
| out | o | Вывод данных в консоль. |
| outl | \_ | Вывод данных в консоль с переносом каретки на новую строку. |
| main | m | Главная функция (точка входа в программу). |
| pow | p | Стандартная функция, возведения в степень целочисленного литерала, языка. |
| abs | a | Стандартная функция, взятия абсолютного значения целочисленного литерала, языка. |
| , | , | Разделитель параметров функции. |
| ; | ; | Признак конца инструкции. |
| { | { | Начало тела функции. |
| } | } | Конец тела функции. |
| ( | ( | Начало перечислений параметров у функции, приоритет операций в выражениях. |
| ) | ) | Конец перечислений параметров у функции, приоритет операций в выражениях. |
| + | + | Оператор сложения. |
| - | - | Оператор вычитания. |
| \* | \* | Оператор умножения. |
| / | / | Оператор деления. |
| % | % | Оператор остатка от деления. |
| > | < | Логический оператор меньше. |
| < | > | Логический оператор больше. |
| == | & | Логический оператор равно. |
| != | ! | Логический оператор не равно. |
| = | = | Знак присваивания. |

Пример графов переходов конечных автоматов соответствующих регулярных выражений изображен на рисунке 3.3.

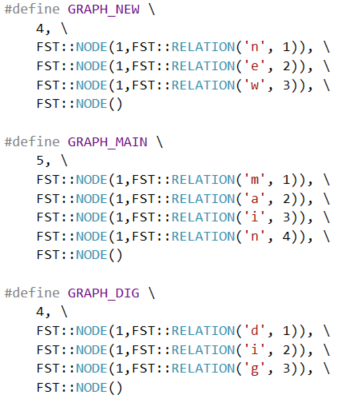


Рисунок 3.3 Фрагмент графов переходов языка PMN-2019

На рисунке 3.4 изображен фрагмент кода функции на языке C++, реализующей алгоритм разбора входной цепочки в соответствии с графами переходов языка PMN-2019.

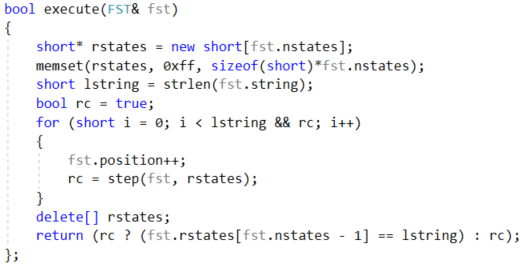


Рисунок 3.4 Функция разбора входной цепочки на языке PMN-2019

* 1. **Основные структуры данных**

Основные структуры данных лексического анализатора: таблица токенов, таблица лексем и таблица идентификаторов.

На рисунке 3.5 изображена реализация структуры таблицы токенов:

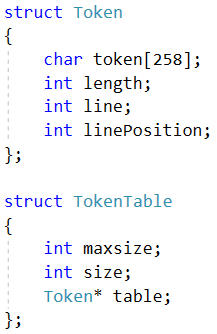


Рисунок 3.5 Структура таблицы токенов языка PMN-2019

Структура TokenTable представляет таблицу токенов, где переменная maxsize хранит число равное максимальному размеру таблицы, size текущий размер таблицы, а table – указатель на строку таблицы.

Структура Token представляет строку таблицы TokenTable, где в массив token записывается само слово, переменная length хранит длину слова, line номер строки в исходном тексте программы, а linePosition – позицию в строке.

На рисунке 3.6 изображена реализация структуры таблицы лексем.

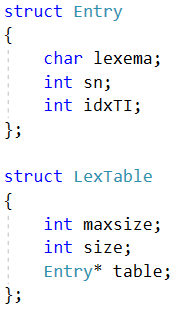


Рисунок 3.6 Структура таблицы лексем языка PMN-2019

Структура LexTable представляет таблицу лексем, где переменная maxsize хранит число равное максимальному размеру таблицы, size текущий размер таблицы, а table – указатель на строку таблицы.

Структура Entry представляет строку таблицы LexTable, где переменная lexeme хранит лексему, sn номер строки в исходном тексте программы, а idxTI – номер в таблице идентификаторов.

На рисунке 3.7 изображена реализация структуры таблицы идентификаторов.

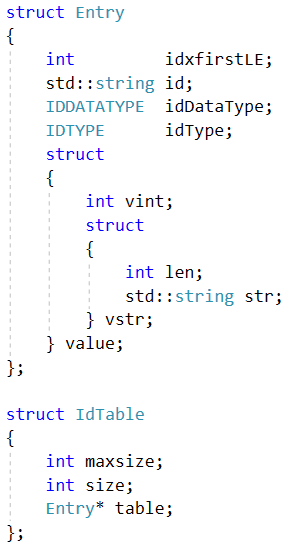


Рисунок 3.7 Структура таблицы идентификаторов языка PMN-2019

Структура IdTable представляет таблицу идентификаторов, где переменная maxsize хранит число равное максимальному размеру таблицы, size текущий размер таблицы, а table – указатель на строку таблицы.

Структура Entry представляет строку таблицы IdTable, где переменная idxfirsttLE хранит индекс первого вхождения в таблицу лексем, id идентификатор, idDataType тип данных, idType тип идентификатора, vint целочисленное значение, len длину строку, а str – строку.

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

При возникновении ошибки в лексическом анализаторе формируется ошибка в следующем формате: Номер ошибки, пояснительный текст, строка в исходном тексте, позиция в строке.

Перечень сообщений лексического анализатора приведен в таблице 3.2.

Таблица 3.2 Перечень сообщений лексического анализатора языка PMN-2019

|  |  |
| --- | --- |
| Номер ошибки | Пояснительный текст |
| 110 | Недопустимый символ в исходном файле (-in:) |
| 111 | Превышена емкость таблицы лексем |
| 112 | Превышено количество строк в таблице лексем |
| 113 | В таблице лексем отсутствует строка с заданным номером |
| 114 | Превышена емкость таблицы идентификаторов |
| 115 | Превышено количество строк в таблице идентификаторов |
| 116 | В таблице идентификаторов отсутствует строка с заданным номером |
| 117 | Не удалось определить тип лексемы |
| 118 | Превышена емкость таблицы токенов |
| 119 | Превышено количество токенов в таблице токенов |
| 120 | Ошибка с разбиением исходного текста на токены |
| 121 | Ошибка с разбором строкового литерала |

* 1. **Принцип обработки ошибок**

При обнаружении ошибки в исходном коде программы лексический анализатор формирует сообщение об ошибки и выводит его в файл с протоколом работы, заданный параметром –log:.

* 1. **Параметры лексического анализатора**

Входным параметром лексического анализа является таблица, состоящая из структур, полями которых являются токен и номер его строки в исходном файле, полученные на этапе проверки исходного кода на допустимость символов.

* 1. **Алгоритм лексического анализа**

Алгоритм лексического анализа языка PMN-2019:

* Считываем исходный текст программы и делим его на токены, формируя структуру таблицы токенов;
* Слова из таблицы токенов пропускаем через графы, определяя тип лексем;
* Составляем таблицу лексем и идентификаторов.

Программный код, реализующий данный алгоритм, изображен на рисунке 3.8.



Рисунок 3.8 Программный код, реализующий лексический анализ

* 1. **Контрольный пример**

Результат работы лексического анализатора, полученный при выполнении контрольного примера, а именно таблица лексем и таблица идентификаторов, представлен в приложении Б.

1. **Разработка синтаксического анализатора**
   1. **Структура синтаксического анализатора**

Синтаксический анализ языка PMN-2019 выполняется после завершения работы лексического анализатора. Синтаксический анализатор предназначен для сопоставления последовательности лексем языка PMN-2019 c его формальной грамматикой.

Входные данные: таблица лексем и таблица идентификаторов.

Результат работы: Дерево разбора.

* 1. **Грамматика, описывающая синтаксис языка**

В синтаксическом анализаторе языка PMN-2019 используется грамматика типа 2 иерархии Хомского (Контекстно-свободная грамматика) G = {N, T, P, S}, где

­- N – конечный алфавит нетерминальных символов, приведенный в первом столбце таблицы 4.1;

- T – конечный алфавит терминальных символов;

- P – конечно множество правил порождения;

- S – начальный нетерминал грамматики G.

Контекстно-свободная грамматика G имеет нормальную форму Грейбах, если она не является леворекурсивной и правила P имеют вид:

1. A → aα, где a T, α N\*;
2. S → λ, где S N – начальный символ, если есть такое правило, то S не должен встречаться в правой части правил.

Алгоритм преобразования грамматик в нормальную форму Грейбах:

1. Исключить недостижимые символы из грамматики;
2. Исключить лямбда-правила из грамматики;
3. Исключить цепные правила.

Перечень и описание терминальных, нетерминальных символов и правил языка приведен в таблице 4.1.

Таблица 4.1 Описание правил, составляющих грамматику языка PMN-2019

|  |  |  |
| --- | --- | --- |
| Нетерминальный символ | Цепочки правил | Описание |
| S | m{N};  m{N};S  dfi(F){N};S  dfi(F){N};  bfi(F){N};S  bfi(F){N};  sfi(F){N};S  sfi(F){N}; | Правила, порождающие главную функцию main и глобальные функции. |
| N | nTi;N  nTi=E;N  nTi;  nTi=E;  i=E;N  i=E;  oL;N  oL;  \_L;N  \_L;  p(W);N  p(W);  a(W);N  a(W);  i(W);N  i(W);  i();N  i();  qQ{N}N  qQ{N}  qQ{N}2{N}N  qQ{N}2{N}  wQ{N}N  wQ{N}  rE;N  rE; | Правила, порождающие конструкции в функциях. |
| F | di  si  bi  di,F  si,F  bi,F | Правила, порождающие параметры объявления функции. |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Нетерминальный символ | Цепочки правил | О |
| E | i  l  iM  lM  (E)M  (E)  i(W)  i()  p(W)  a(W)  i(W)M  i()M  p(W)M  a(W)M | Правила, порождающие выражения. |
| W | i  l  i,W  l,W | Правила, порождающие параметры вызываемой функции. |
| M | +E  -E  \*E  /E  %E  +EM  -EM  \*EM  /EM  %EM  !L  &L  >L  <L | Правила, порождающие операторы. |
| Q | (L<L)  (L>L)  (L!L)  (L&L) | Правила, порождающие условия в условных конструкциях. |
| L | l  i | Правила, порождающие литерал и идентификатор. |
| T | d  s  b | Правила, порождающие типы данных. |

* 1. **Построение конечного магазинного автомата**

Конечный автомат с магазинной памятью представляет собой семерку, описание которой приведено ниже.

M = {Q, V, Z, δ, q0, z0, F}, где

- Q – множество состояний;

- V – алфавит входных символов;

- Z – специальный алфавит магазинных символов;

- δ – функция переходов автомата;

- q0 Q – начальное состояние автомата;

- z0 Z – начальное состояние магазина (маркер дна);

- F Q – Множество конечных состояний.

Схема конечного автомата с магазинной памятью изображена на рисунке 4.1.

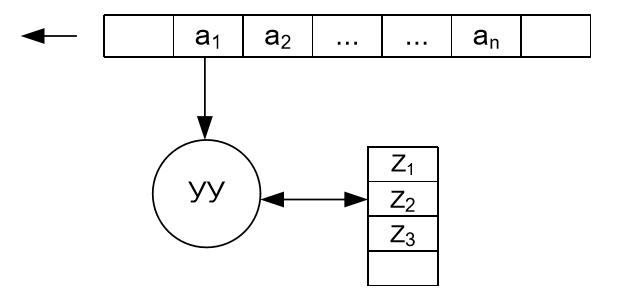


Рисунок 4.1 Схема конечного автомата с магазинной памятью

Алгоритм работы конечного автомата с магазинной памятью:

1. Состояние автомата (q, aα, zβ);
2. Читает символ а находящийся под головкой (сдвигает ленту);
3. Не читает ничего (читает λ, не сдвигает ленту);
4. Из δ определяет новое состояние q', если (q', γ) δ(q, a, z) или

(q', γ) δ(q, λ, z);

1. Читает верхний (в стеке) символ z и записывает цепочку γ т.к.

(q', γ) δ(q, a, z), при этом, если γ = λ, то верхний символ магазина просто удаляется;

1. Работа автомата заканчивается (q, λ, λ).

Последовательность мгновенных состояний конечного автомата с магазинной памятью, демонстрирующая успешный разбор цепочки из контрольного примера приведена в приложении В.

* 1. **Основные структуры данных**

Программный код основных структур данных на языке C++, описывающих контекстно-свободную грамматику представлен в приложении Г.

* 1. **Описание алгоритма синтаксического разбора**

Алгоритм синтаксического разбора языка PMN-2019:

1. В магазин заноситься стартовый символ;
2. Формируется входная лента, полученная из таблицы лексем;
3. Нетерминальный символ раскрывается, согласно правилам, и записывается в магазин;
4. Если терминал на вершине стека и в начале ленты совпадают, то данный терминал удаляется из входной ленты. Иначе возвращается в предыдущее состояние и выбирает другую цепочку нетерминала;
5. Если в магазине встречается нетерминал, переход к пункту 3;
6. Если достигнуто дно стека и входная цепочка пуста, то синтаксический анализ выполнен успешно. Иначе генерируется исключение.

Обобщенная блок-схема алгоритма синтаксического анализа изображена на рисунке 4.2.

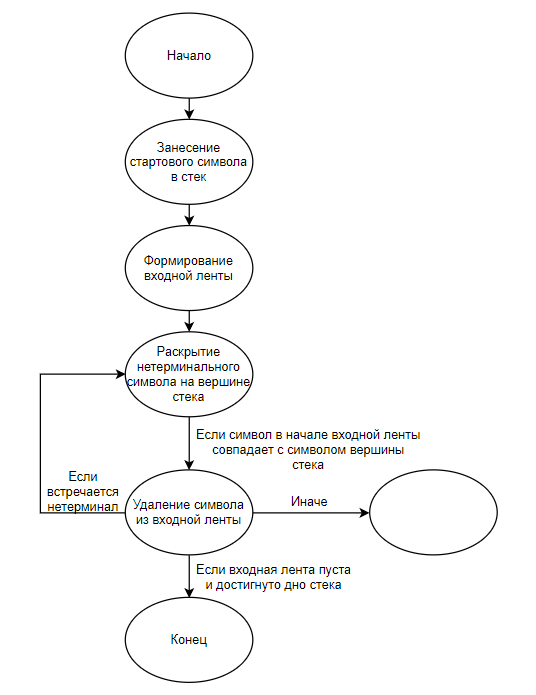


Рисунок 4.2 Блок-схема алгоритма синтаксического анализа

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

При возникновении ошибки в синтаксическом анализаторе формируется ошибка в следующем формате: Номер ошибки, пояснительный текст, строка в исходном тексте.

Перечень сообщений синтаксического анализатора приведен в таблице 4.2.

Таблица 4.2 Перечень сообщений синтаксического анализатора языка PMN-2019

|  |  |
| --- | --- |
| Номер ошибки | Пояснительный текст |
| 600 | Неверная структура программы |
| 601 | Ошибка в конструкции функции |
| 602 | Ошибка в выражении |
| 603 | Ошибка в параметрах функции |
| 604 | Ошибка в параметрах вызываемой функции |
| 605 | Ошибочный оператор |
| 606 | Ошибочное условие в условной конструкции |
| 607 | Ошибка типа |

* 1. **Параметры синтаксического анализатора и режимы его работы**

Входными параметром синтаксического анализатора являются таблицы лексем и идентификаторов, состоящие из структур, полями которых являются лексемы и идентификаторы, полученные на этапе лексического анализа.

* 1. **Принцип обработки ошибок**

При обнаружении ошибки в исходном коде программы синтаксический анализатор формирует сообщение об ошибки и выводит его в файл с протоколом работы, заданный параметром –log:.

* 1. **Контрольный пример**

Результат работы синтаксического анализатора, полученный при выполнении контрольного примера, а именно дерево разбора, представлен в приложении В.

1. **Разработка семантического анализатора**
   1. **Структура семантического анализатора**

Семантический анализ языка PMN-2019 выполняется после выполнения лексического и синтаксического анализа. Несмотря на это, некоторые семантические проверки выполняются на этапе лексического анализа. На вход семантического анализатора подаются таблица лексем и таблица идентификаторов. Схема семантического анализатора изображена на рисунке 5.1.

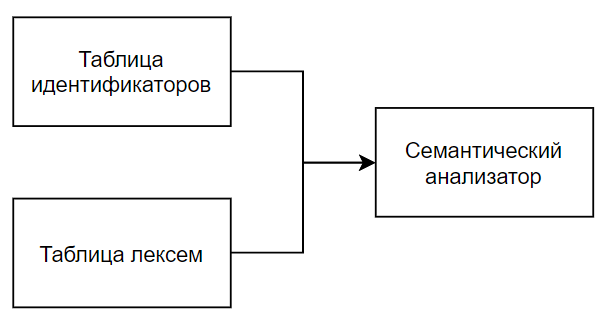


Рисунок 5.1 Схема семантического анализатора языка PMN-2019

* 1. **Функции семантического анализатора**

Семантические проверки языка PMN-2019 с указанием фаз их выполнения приведены в таблице 5.1.

Таблица 5.1 Семантические проверки языка PMN-2019

|  |  |
| --- | --- |
| Семантическая проверка | Фаза выполнения |
| Превышение длины строки | Лексический анализ |
| Превышение целочисленного значения | Лексический анализ |
| Повторная реализация функции main | Лексический анализ |
| Превышение длины лексемы | Лексический анализ |
| Наличие точки входа в программу | Лексический анализ |
| Объявление идентификатора перед использованием | Семантический анализ |
| Повторное объявление идентификатора | Семантический анализ |
| Соответствие типов в выражении | Семантический анализ |
| Количество параметров функции | Семантический анализ |
| Соответствие параметров объявленной и вызываемой функции | Семантический анализ |
| Соответствие параметров встроенной функции | Семантический анализ |

Продолжение таблицы 5.1

|  |  |
| --- | --- |
| Семантическая проверка | Фаза выполнения |
| Проверка левосторонних выражений | Семантический анализ |
| Повторная реализация функции | Семантический анализ |
| Соответствие типа возвращаемого значения типу функции | Семантический анализ |
| Перегрузка оператора для работы со строками | Семантический анализ |

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

Структура и текст сообщений семантического анализатора приведены в таблице 5.2.

Таблица 4.2 Перечень сообщений семантического анализатора языка PMN-2019

|  |  |
| --- | --- |
| Номер ошибки | Пояснительный текст |
| 130 | Превышена длина строк в 255 символов. |
| 131 | Функция main уже имеет реализацию. |
| 132 | Превышена длина лексемы. |
| 133 | Превышено значение INT (1 byte). |
| 134 | Не найдена точка входа в программу (main). |
| 135 | Идентификатор с таким именем не найден. |
| 136 | Повторное объявление идентификатора. |
| 137 | Несоответствие типов в выражении. |
| 138 | Слишком много параметров в функции. |
| 139 | Превышено количество функций. |
| 140 | Несоответствие параметров объявленной и вызываемой функции. |
| 141 | Несоответствие параметров встроенной функции. |
| 142 | Идентификатор не является левосторонним выражением. |
| 143 | Данная функция уже имеет реализацию. |
| 144 | В вызове функции отсутствуют (). |
| 145 | Тип возвращаемого значения не соответствует типу функции. |
| 146 | Оператор не перегружен для работы со строками. |

* 1. **Принцип обработки ошибок**

При обнаружении ошибки в исходном коде программы семантический анализатор формирует сообщение об ошибки и выводит его в файл с протоколом работы, заданный параметром –log:.

* 1. **Контрольный пример**

Контрольный пример для демонстрации ошибок, диагностируемых семантическим анализатором вместе с отчетом выданных сообщений представлен в приложении.

1. **Вычисление выражений**
   1. **Выражение, допускаемые языком**

В языке PMN-2019 допускаются выражения, применимые к целочисленным типом данным. Допускается использование функций в выражениях. Операции и их приоритетность приведена в таблице 6.1.

Таблица 5.1 Операции и их приоритетность языка PMN-2019

|  |  |
| --- | --- |
| Приоритет | Операция |
| 0 | ( |
| 0 | ) |
| 1 | , |
| 2 | + |
| 2 | - |
| 3 | \* |
| 3 | / |
| 3 | % |

Примеры выражений из контрольного примера:

(pow(abs(-5), 4) - 4) \* 2, (24 % (2\*5)), 2+5/5 и т.п.

* 1. **Польская запись и принцип её построения**

Язык PMN-2019 транслируется в язык ассемблера, в котором все вычисления производятся через стек. Преобразование исходных выражений в обратную польскую запись, упрощает генерацию кода вычисления выражений в язык ассемблера. Алгоритм построения польской записи приведен ниже:

Пока есть символы для чтения:

- Читаем очередной символ;

- Если символ является числом, добавляем его к выходной строке;

- Если символ является функцией, помещаем его в стек;

- Если символ является открывающей скобкой, помещаем его в стек;

- Если символ является закрывающей скобкой:

До тех пор, пока верхним элементом стека не станет открывающая скобка, выталкиваем элементы из стека в выходную строку. При этом открывающая скобка удаляется из стека, но в выходную строку не добавляется.

- Если символ является операцией:

1) Пока операция на вершине стека приоритетнее или пока на вершине стека функция;

2) Помещаем операцию в стек.

Когда входная строка закончилась, выталкиваем все символы из стека в выходную строку.

* 1. **Программная реализация обработки выражений**

Фрагмент кода, реализующего преобразование выражений в обратный польский формат изображен на рисунке 6.1.

При встрече лексемы идентификатора, идет проверка на функцию. Если идентификатор является функцией, идентификатор помещается в стек. Иначе идентификатор помещается в выходную строку.

При встрече литерала, литерал помещается в выходную строку.

При встрече открывающей скобки, скобка кладется в стек.

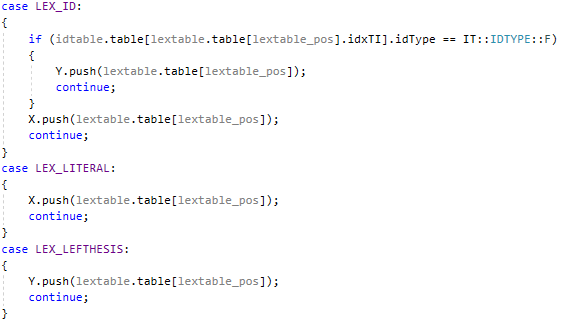


Рисунок 6.1 Фрагмент кода, реализующего преобразование выражений

* 1. **Контрольный пример**

В приложении Б представлена обновленная таблица лексем, с выражениями, приведенными к обратной польской записи.

1. **Генерация кода**
   1. **Структура генератора кода**

Трансляция с языка PMN-2019 производиться в язык ассемблера. Структура генератора кода изображена на рисунке 7.1.

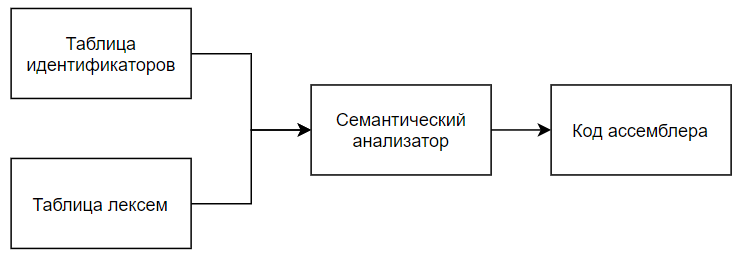


Рисунок 7.1 Структура генератора кода

Генератор кода последовательно проходит таблицу лексем, при необходимости обращаясь к таблице идентификаторов. В зависимости от пройденных лексем выполняется генерация кода ассемблера.

* 1. **Представление типов данных в оперативной памяти**

Плоская модель памяти (flat): приложению для кода и данных предоставляется один непрерывный сегмент. Данный сегмент в свою очередь разбит на страницы:

- .STACK – стек;

- .CONST – константы;

- .DATA – переменные;

- .CODE – код.

Соответствие типов данных в исходном языке программирования PMN-2019 типам целевого языка приведены в таблице 7.1.

Таблица 6.1 Соответствие типов языка PMN-2019 типам языка ассемблера

|  |  |  |
| --- | --- | --- |
| Тип данных языка PMN-2019 | Тип данных ассемблера | Пояснение |
| dig | SDWORD | Хранит знаковый целочисленный тип. |
| str | DWORD | Хранит указатель на начало строки. |
| bool | DWORD | Хранит булево значение. |

* 1. **Статическая библиотека**

Функции, входящие в состав статической библиотеки языка PMN-2019, приведены в таблице 1.7.

Статическая библиотека написана на языке C++. Подключение статической библиотеки производится на этапе генерации кода.

* 1. **Особенности алгоритма генерации кода**

Обобщенная блок-схема алгоритма генерации кода языка ассемблера изображена на рисунке 7.2.

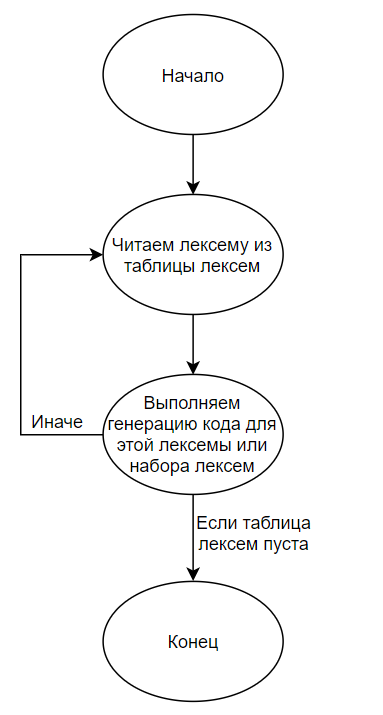


Рисунок 7.2 Блок-схема алгоритма генерации кода языка ассемблер

Пока таблица лексем не пуста, читаем лексему:

- Если для лексемы есть код генерации, генерируем код;

- Иначе читаем следующую лексему.

* 1. **Входные параметры генератора**

Входными параметрами генератора являются таблица идентификаторов и таблица лексем, которые предназначены для генерации кода ассемблера.

* 1. **Контрольный пример**

Результат генерации кода на основе контрольного примера представлен в приложении Д.

1. **Тестирование транслятора**
   1. **Общие положения**

Тесты, включенные в контрольный пример, отражают общую работу транслятора: корректную трансляцию, учет проверок и генерация ошибки при возникновении исключения.

При возникновении ошибки, работа компилятора будет приостановлена. Текст с номером и сообщением об ошибки будет выведен в файл, заданный параметром –log:.

* 1. **Результаты тестирования**

Описание тестовых наборов, демонстрирующих проверки на разных этапах трансляции, приведено в таблице 8.1.

Таблица 8.1 Описание тестовых наборов языка PMN-2019

|  |  |  |  |
| --- | --- | --- | --- |
| Фрагмент исходного кода | Этап | Код ошибки | Текст сообщения |
| new dig gf#; | Лексический анализ | 110 | Недопустимый символ в исходном файле |
| 24fdfg str stroka; | Лексический анализ | 117 | Не удалось определить тип лексемы |
| main {} | Синтаксический анализ | 600 | Неверная структура программы |
| if (5 > 6) { | Синтаксический анализ | 601 | Ошибка в конструкции функции |
| x = 5 ++ 6; | Синтаксический анализ | 602 | Ошибка в выражении |
| string function strk(str a,) {}; | Синтаксический анализ | 603 | Ошибка в параметрах функции |
| new dig x = pow(2,); | Синтаксический анализ | 604 | Ошибка в параметрах вызываемой функции |
| x = (20 – 5; | Синтаксический анализ | 605 | Ошибочный оператор |
| if (5 > x + 2) | Синтаксический анализ | 606 | Ошибочное условие в условной конструкции |
| main{};  main{}; | Семантический анализ | 131 | Функция main уже имеет реализацию |
| new dig x = 244; | Семантический анализ | 133 | Превышено значение целочисленного литерала |

Продолжение таблицы 8.1

|  |  |  |  |
| --- | --- | --- | --- |
| Фрагмент исходного кода | Этап | Код ошибки | Текст сообщения |
| main  {  x = 5;  …  }; | Семантический анализ | 135 | Идентификатор с таким именем не найден |
| main  {  new dig x;  new dig x;  …  }; | Семантический анализ | 136 | Повторное объявление идентификатора |
| main  {  new dig x = 5;  new str y = “stroka”;  x = x + y;  …  }; | Семантический анализ | 137 | Несоответствие типов в выражении |
| dig function fun(str x)  {  …  fun(5);  }; | Семантический анализ | 140 | Несоответствие параметров объявленной и вызываемой функции |
| dig function fun(str x)  {  …  };  main  {  fun = 5;  …  }; | Семантический анализ | 142 | Идентификатор не является левосторонним выражением |
| dig function fun(str x)  {  …  };  str function fun(dig x)  … | Семантический анализ | 143 | Данная функция уже имеет реализацию |

# **Заключение**

В ходе выполнения курсовой работы был разработан транслятор языка PMN-2019 в язык ассемблера и написана пояснительная записка со спецификацией языка. Выполнены минимальные требования к курсовому проекту, а так же ряд дополнительных, повышающий бал, требований.

Количественные и качественные характеристики реализации транслятора:

* Количество типов данных: 3;
* Количество инструкция языка: 7;
* Количество лексем: 32;
* Правил грамматики: 81;
* Наличие стандартной библиотеки;
* Количество строк на языке реализации: 3563;
* Время трансляции контрольного примера: 0.29 секунды;

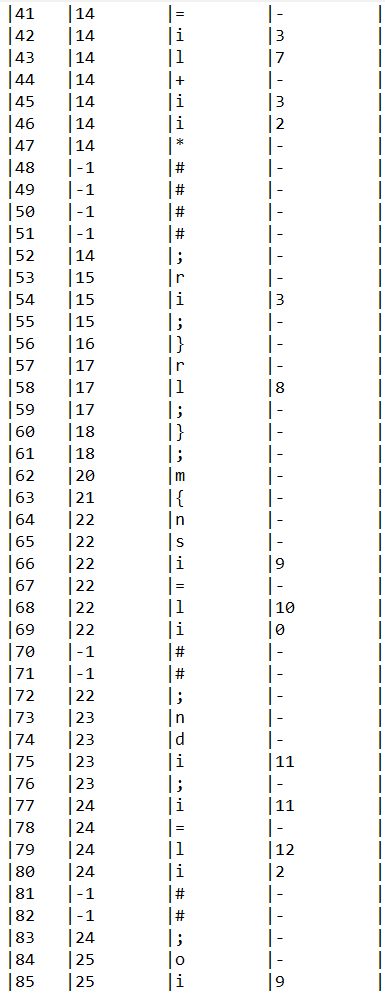
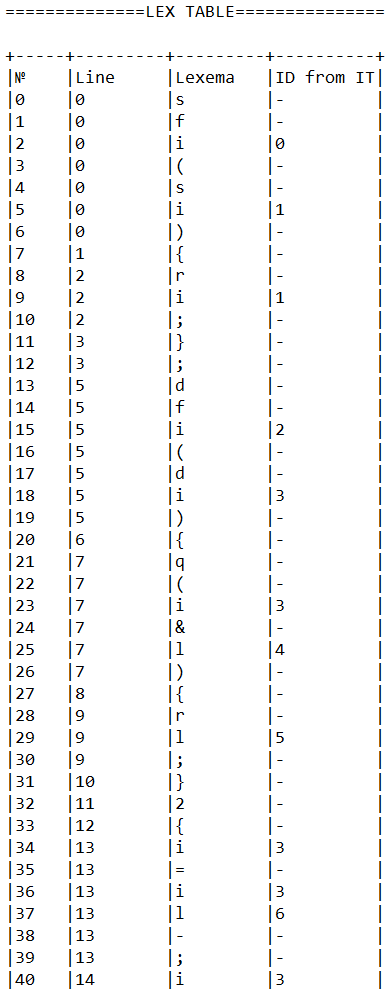
По итогу выполнения курсового проекта по дисциплине «Языки программирования» были приобретены навыки разработки системы программирования (трансляторов, интерпретаторов), изучены основы теории формальных грамматик и основы общей теории компиляторов.

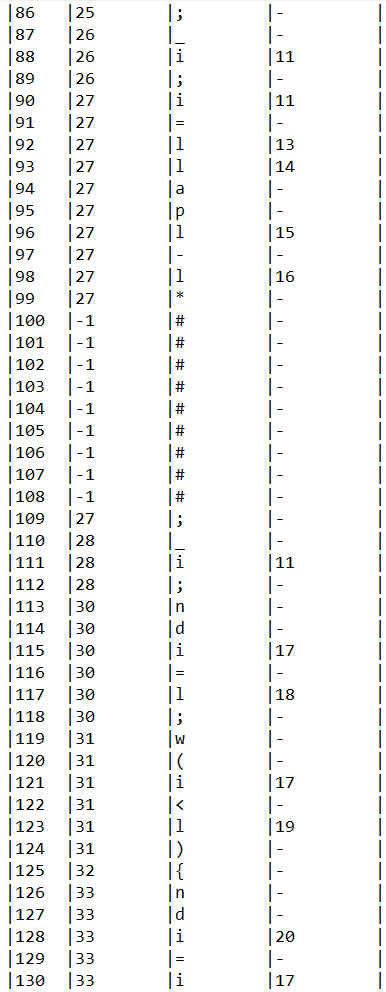
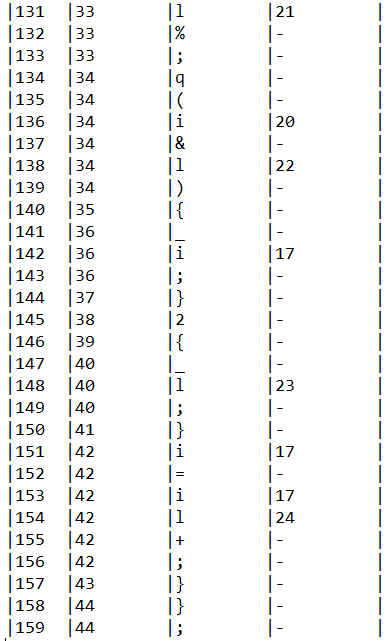
# **Список использованных источников**

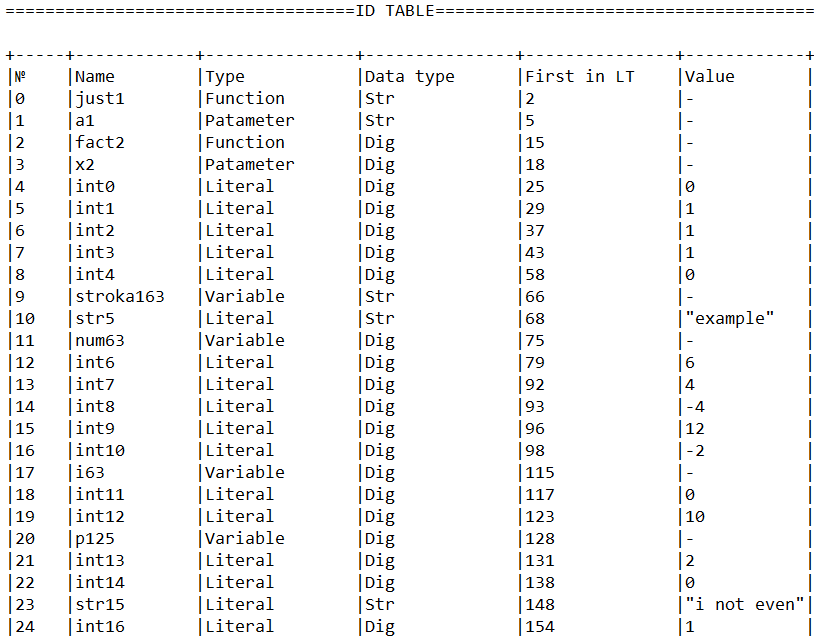
# **Приложение А**

|  |
| --- |
| str function just(str a)  {  return a;  };  dig function fact(dig x)  {  if (x == 0)  {  return 1;  }  else  {  x = x - 1;  x = (x + 1) \* fact(x);  return x;  }  return 0;  };  main  {  new str stroka1 = just("example");  new dig num;  num = fact(6);  out stroka1;  outl num;  num = ((pow(4, abs(-4)) -12) \* -2 );  outl num;    new dig i = 0;  while (i < 10)  {  new dig p = i % 2;  if (p==0)  {  outl i;  }  else  {  outl "i not even";  }  i = i + 1;  }  }; |

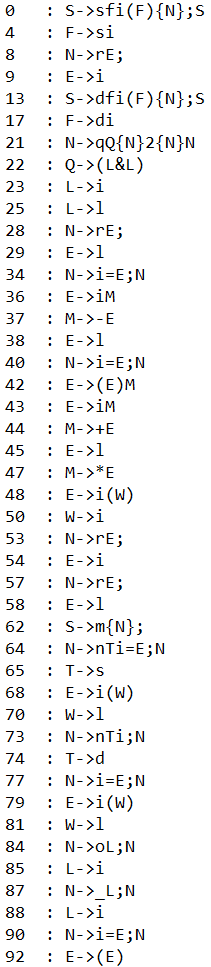
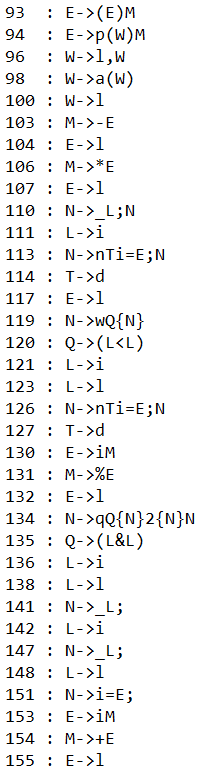
# **Приложение Б**



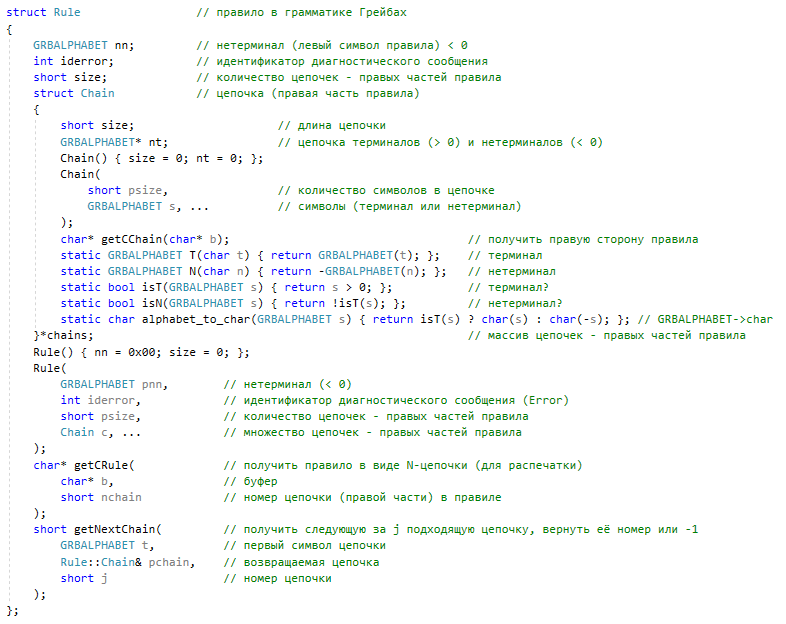


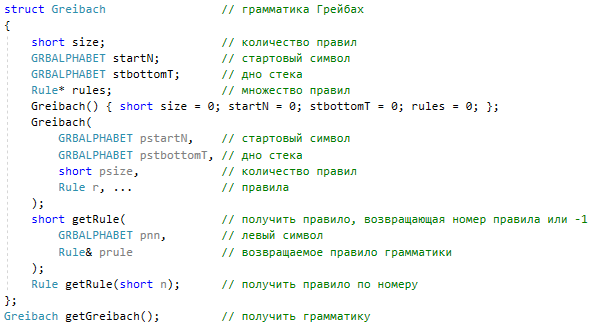


# **Приложение В**



# **Приложение Г**





# **Приложение Д**

|  |
| --- |
| .586  .model flat, stdcall  includelib libucrt.lib  includelib kernel32.lib  includelib "..\Debug\PMN\_LIB.lib"  ExitProcess PROTO:DWORD  SYSPAUSE PROTO  nout PROTO : SDWORD  sout PROTO : BYTE  soutl PROTO : BYTE  noutl PROTO : SDWORD  power PROTO : SDWORD, : SDWORD  abser PROTO : SDWORD  .STACK 4096  .CONST  null\_division BYTE 'ERROR: DIVISION BY ZERO', 0  true BYTE 'true', 0  false BYTE 'false', 0  int0 SDWORD 0  int1 SDWORD 1  int2 SDWORD 1  int3 SDWORD 1  int4 SDWORD 0  str5 BYTE "example", 0  int6 SDWORD 6  int7 SDWORD 4  int8 SDWORD -4  int9 SDWORD 12  int10 SDWORD -2  int11 SDWORD 0  int12 SDWORD 10  int13 SDWORD 2  int14 SDWORD 0  str15 BYTE "i not even", 0  int16 SDWORD 1  .DATA  stroka163 DWORD ?  num63 SDWORD 0  i63 SDWORD 0  p125 SDWORD 0  .CODE  just1 PROC a1 : DWORD  push a1  jmp local0  local0:  pop eax  ret  just1 ENDP  fact2 PROC x2 : SDWORD  mov eax, x2  cmp eax, int0  jz ifi1  jnz else1  ifi1:  push 1  jmp local1  jmp ifEnd1  else1:  push x2  push int2  pop ebx  pop eax  sub eax, ebx  push eax  pop x2  push x2  push int3  pop eax  pop ebx  add eax, ebx  push eax  push x2  call fact2  push eax  pop eax  pop ebx  mul ebx  push eax  pop x2  push x2  jmp local1  ifEnd1:  push 0  jmp local1  local1:  pop eax  ret  fact2 ENDP  main PROC  push offset str5  call just1  push eax  pop stroka163  push int6  call fact2  push eax  pop num63  push stroka163  call sout  push num63  call noutl  push int7  push int8  call abser  push eax  call power  push eax  push int9  pop ebx  pop eax  sub eax, ebx  push eax  push int10  pop eax  pop ebx  mul ebx  push eax  pop num63  push num63  call noutl  push int11  pop i63  while1:  mov eax, i63  cmp eax, int12  jl whileT1  jge whileEnd1  whileT1:  push i63  push int13  pop ebx  pop eax  cmp ebx,0  je SOMETHINGWRONG  cdq  idiv ebx  push edx  pop p125  mov eax, p125  cmp eax, int14  jz ifi1  jnz else1  ifi1:  push i63  call noutl  jmp ifEnd1  else1:  push offset str15  call soutl  ifEnd1:  push i63  push int16  pop eax  pop ebx  add eax, ebx  push eax  pop i63  jmp while1  whileEnd1:  call SYSPAUSE  push 0  call ExitProcess  SOMETHINGWRONG:  push offset null\_division  call soutl  call SYSPAUSE  push -1  call ExitProcess  main ENDP  end main |