МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 «Программное обеспечение информационных технологий»

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора GRI-2018 »

Выполнил студент Грунковский Роман Иванович  (Ф.И.О.)

Руководитель проекта ст.пр. Наркевич Аделина Сергеевна  (учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Пацей Н.В.  (учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты ст.пр. Наркевич Аделина Сергеевна  (учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер ст.пр. Наркевич Аделина Сергеевна  (учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2018

Оглавление

[Введение 5](#_Toc533111982)

[1. Спецификация языка программирования 6](#_Toc533111983)

[1.1. Характеристика языка программирования 6](#_Toc533111984)

[1.2. Алфавит языка 6](#_Toc533111985)

[1.3. Применяемые сепараторы 6](#_Toc533111986)

[1.4. Применяемые кодировки 6](#_Toc533111987)

[1.5. Типы данных 7](#_Toc533111988)

[1.6. Преобразование типов данных 8](#_Toc533111989)

[1.7. Идентификаторы 8](#_Toc533111990)

[1.8. Литералы 8](#_Toc533111991)

[1.9. Объявление данных и область видимости 8](#_Toc533111992)

[1.10. Инициализация данных 8](#_Toc533111993)

[1.11. Инструкция языка 9](#_Toc533111994)

[1.12. Операции языка 9](#_Toc533111995)

[1.13. Выражения и их вычисления 9](#_Toc533111996)

[1.14. Программные конструкции языка 10](#_Toc533111997)

[1.15. Область видимости идентификаторов 10](#_Toc533111998)

[1.16. Семантические проверки 10](#_Toc533111999)

[1.17. Распределение оперативной памяти на этапе выполнения 11](#_Toc533112000)

[1.18. Стандартная библиотека и её состав 11](#_Toc533112001)

[1.19. Ввод и вывод данных 12](#_Toc533112002)

[1.20. Точка входа 12](#_Toc533112003)

[1.21. Препроцессор 12](#_Toc533112004)

[1.22. Соглашения о вызовах 12](#_Toc533112005)

[1.23. Объектный код 12](#_Toc533112006)

[1.24. Классификация сообщений транслятора 13](#_Toc533112007)

[1.25. Контрольный пример 13](#_Toc533112008)

[2. Структура транслятора 14](#_Toc533112009)

[2.1. Компоненты транслятора их назначение и принципы взаимодействия 14](#_Toc533112010)

[2.2. Перечень параметров транслятора входных 15](#_Toc533112011)

[2.3. Перечень протоколов формируемых транслятором и их содержимое 15](#_Toc533112012)

[3. Разработка лексического анализатора 17](#_Toc533112013)

[3.1. Структура лексического анализатора 17](#_Toc533112014)

[3.2. Контроль входных символов 17](#_Toc533112015)

[3.3. Удаление избыточных символов 18](#_Toc533112016)

[3.4. Перечень ключевых слов, сепараторов, символов операций, соответствующих им лексем и конечных автоматов 18](#_Toc533112017)

[3.5. Основные структуры данных 19](#_Toc533112018)

[3.6. Принцип обработки ошибок и их перечень 19](#_Toc533112019)

[3.7. Структура и перечень сообщений лексического анализатора 20](#_Toc533112020)

[3.8. Параметры лексического анализатора и режимы его работы 20](#_Toc533112021)

[3.9. Алгоритм лексического анализатора 20](#_Toc533112022)

[3.10. Контрольный пример 21](#_Toc533112023)

[4. Разработка синтаксического анализатора 22](#_Toc533112024)

[4.1. Структура синтаксического анализатора 22](#_Toc533112025)

[4.2. Грамматика языка 22](#_Toc533112026)

[4.3. Построение конечного магазинного автомата 23](#_Toc533112027)

[4.4. Основные структуры данных 24](#_Toc533112028)

[4.5. Описание алгоритма синтаксического разбора 24](#_Toc533112029)

[4.6. Структура и перечень сообщений синтаксического анализатора 24](#_Toc533112030)

[4.7. Параметры синтаксического анализатора и режимы его работы 25](#_Toc533112031)

[4.8. Принцип обработки ошибок 25](#_Toc533112032)

[4.9. Контрольный пример 25](#_Toc533112033)

[5. Разработка семантического анализатора 26](#_Toc533112034)

[5.1. Структура семантического анализатора 26](#_Toc533112035)

[5.2. Функции семантического анализатора 26](#_Toc533112036)

[5.3. Принцип обработки ошибок 26](#_Toc533112037)

[5.4. Контрольный пример 27](#_Toc533112038)

[6. Преобразование выражений 28](#_Toc533112039)

[6.1. Выражения, допускаемые языком 28](#_Toc533112040)

[6.2. Польская запись и принцип её построения 28](#_Toc533112041)

[6.3. Примеры преобразования выражений 29](#_Toc533112042)

[7. Генерация кода 30](#_Toc533112043)

[8. Тестирование транслятора 32](#_Toc533112044)

[Заключение 33](#_Toc533112045)

[Литература 34](#_Toc533112046)

[Приложение А: Таблица входных символов. 35](#_Toc533112047)

[Приложение Б: Графы переходов 36](#_Toc533112048)

[Приложение Г: Состояния автомат с магазинной памятью 39](#_Toc533112049)

[Приложение Д: Автомат с магазинной памятью 40](#_Toc533112050)

[Приложение Е: Дерево разбора 41](#_Toc533112051)

[Приложение Ж: Тестирование семантического анализатора 43](#_Toc533112052)

[Приложение И: Контрольный пример. 44](#_Toc533112053)

# Введение

В данном курсовом проекте была поставлена задача разработать собственный язык программирования, а также транслятор для него. Язык получил название GRI-2018.

Задание на курсовой проект можно разделить на следующие подзадания:

* Разработка спецификации для языка GRI-2018 (глава 1);
* Разработка структуры транслятора (глава 2);
* Разработка лексического анализатора (глава 3);
* Разработка синтаксического анализатора (глава 4);
* Разработка семантического анализатора (глава 5);
* Преобразование выражений (глава 6);
* Генерация кода (глава 7);
* Тестирование транслятора (глава 8);

# 1. Спецификация языка программирования

# 1.1. Характеристика языка программирования

Язык программирования GRI-2018 является процедурным, универсальным, строго типизированным, не объектно-ориентированным, компилируемый. Язык допускает использование 2 типов данных: строковый и целочисленный. Все переменные являются локальными. Для работы с целочисленными данными разрешено использование деления, умножения, вычитания и сложения.

# 1.2. Алфавит языка

Символы, разрешенные к использованию при написании кода: латинские символы размером один байт, кодировки Windows-1251. Данная таблица приведена на рисунке 1. Русский язык запрещён. В литералах, разрешено использовать десятичные цифры и алфавит латиницы.

Символы, используемые на этапе выполнения: [a…z], [A…Z], [0…9]. А также спецсимволы: { } ( ) ; \* + - / = , “ пробел.

# 1.3. Применяемые сепараторы

В языке GRI-2018 используются сепараторы, представленные в табл. 1.1*.* Пробел допускается везде кроме идентификаторов и ключевых слов.

Таблица 1.1. Сепараторы, применяемые в языке GRI-2018

|  |  |
| --- | --- |
| Сепаратор | Описание |
| ; - точка с запятой | Разделитель конструкций |
| {} - фигурные скобки | Программный блок |
| () - круглые скобки. | Приоритетность операций |
| , - запятая | Разделитель параметров функции |

Получаем, что сепараторы данного языка совпадают с сепараторами схожего по синтаксису языка C++.

# 1.4. Применяемые кодировки

Кодировка, используемая для написания программ на языке GRI-2018, - стандартная кодировка Windows-1251 представленная на рисунке 1.

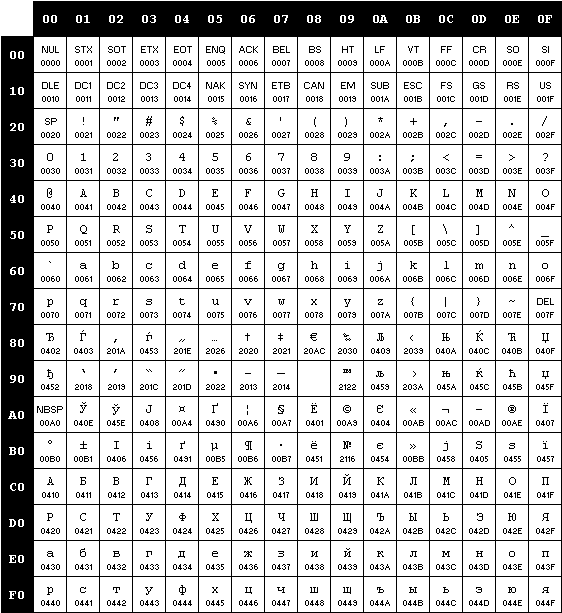


Рисунок 1, Основная таблица Windows-1251

# 1.5. Типы данных

В данном языке можно использовать 2 типа данных, которые описаны в таблице 1.2.

Таблица 1.2, Основные типы данных языка GRI-2018

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Тип данных | Краткое описание | Занимаемое место в памяти | Границы | Значение при инициализации: |
| int | Целое значение | 4 байта | От -65536 до 65536. | 0 |
| str | Строка | 256 байт | От 1 до 255  разрешенных символов | Пустая строка, длина строки 0 |

# 1.6. Преобразование типов данных

В языке GRI-2018 преобразование данных не предусмотрено.

# 1.7. Идентификаторы

В языке GRI-2018 идентификаторы должны быть составлены только из символов нижнего регистра английского алфавита. Типы идентификаторов: имя переменной или функции, литерал, параметр, имя стандартной функции. Идентификатор составляется из букв английского алфавита от 1 до 9 символов, без пробелов. Максимальная длина идентификатора равна 9 символам, в противном случае урезается до указанной длины. Идентификатор не может совпадать с ключевыми словами.

Пример:

value int a – Правильно.

value int function– Не правильно.

value int glavn – Не правильно.

# 1.8. Литералы

Таблица 1.3, Литералы в языке GRI-2018

|  |  |  |
| --- | --- | --- |
| Тип литерала | Допустимые символы | Допустимые значения: |
| int | Цифры от 0 до 9 | От 0 до 65536 |
| str | Разрешенные символы | От 1 до 255 символов |

Строковый тип – комбинация символов, которая может содержать символы латинского алфавита, а также символы пробела, цифр, специальных символов (исключая символ двойных кавычек), ограниченных с помощью двойных кавычек с обеих сторон.

Целочисленный тип – комбинация из цифр от 0 до 9.

# 1.9. Объявление данных и область видимости

В языке GRI-2018 объявление данных начинается с ключевого слова value, указывается тип данных и имя идентификатора.

Пример: value int a, value str b;

Область видимости: сверху вниз, параметры внутри функции, объявления внутри функции видны только внутри функции, объявления переменных за пределами функций и главной функции не предусмотрены.

# 

# 1.10. Инициализация данных

Инициализация переменной происходит после её объявления. Инициализация переменной в момент объявления запрещена.

Например: value str a; a = “word”; value int b; b = 5;

# 1.11. Инструкция языка

В языке GRI-2018 предусмотрены следующие инструкции, представленные в таблице 1.4.

Таблица 1.4, Инструкции языка GRI-2018

|  |  |
| --- | --- |
| Инициализация переменной | Имя переменной = значение; |
| Создание внешней функции | function идентификатор (тип данных идентификатор, …)  Область видимости сверху вниз (по принципу С++). Все переменные являются локальными. |
| Главная функция | glavn  {  …  } |
| Возврат из подпрограммы | back идентификатор или литерал; |
| Вывод данных | out(идентификатор); |

# 1.12. Операции языка

В языке GRI-2018 предусмотрены следующие операции с данными. Приоритетность операций определяется с помощью (). Операции представлены в таблице 1.5.

Таблица 1.5, Операции языка GRI-2018

|  |  |
| --- | --- |
| Операция | Описание |
| + | бинарный, арифметическое суммирование, (int + int); |
| - | бинарный, арифметическое вычитание, (int - int); |
| \* | бинарный, арифметическое умножение, (int \* int); |
| / | бинарный, арифметическое целочисленное деление, (int / int); |

# 1.13. Выражения и их вычисления

Вычисление выражений осуществляется по некоторым правилам, которые приведены ниже:

1. Выражение записываются в одну строку;
2. Допускается использовать скобки (приоритет);
3. Использование двух операторов подряд не допускается;
4. Выражение может содержать вызов функции;

Пример:

int function sum (int a. int b)

{..}

value int new;

new=sum(1,2)+20;

# 1.14. Программные конструкции языка

В языке программирования GRI-2018 предусмотрена одна главная функция и внешние функции. Программные конструкции представлены в таблице 1.6.

Таблица 1.6, Программные конструкции GRI-2018

|  |  |
| --- | --- |
| Внешняя функция | Тип данных function идентификатор (тип данных идентификатор, … )  {  …  back идентификатор | литерал;  }  Область видимости сверху вниз (по принципу С++). |
| Главная функция | glavn  {  |программный блок|  }  Область видимости сверху вниз. Все переменные являются локальными. |

# 1.15. Область видимости идентификаторов

Сверху вниз, параметры внутри функции, объявления внутри функции видны только внутри функции, объявления за пределами функций и главной функции запрещены.

# 1.16. Семантические проверки

Основные семантические правила, проверяемые на этапах работы транслятора, представлены в таблице 1.7. Набор правил, приведенный в данной таблице, облегает дальнейшую обработку программы, написанной на языке GRI-2018.

Таблица 1.7, Семантические правила

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Проверка на наличие точки входа в программу main() |
| 2 | Тип параметров функции в объявлении и в вызове должны совпадать. |
| 3 | Не допускаются идентификаторы, совпадающие с ключевыми словами. |
| 4 | Названия функций не должны повторяться. |
| 5 | Нет повторяющихся объявлений идентификаторов в одной области видимости. |
| 6 | Переменные, используемые в функции, должны быть предварительно объявлены, после чего инициализированы. |
| 7 | Тип возвращаемого значения и тип функции должны соответствовать. |
| 8 | Проверка на максимальное и минимальное значение целочисленного литерала. |
| 9 | Проверка на максимальную длину строкового литерала. |
| 10 | Проверка на то, присвоено ли значение переменной во время её объявления. |
| 11 | Проверка деления на 0. |

# 1.17. Распределение оперативной памяти на этапе выполнения

Транслированный код использует 2 области памяти.

В область констант заносятся литералы.

В область глобальных переменных заносятся переменные и параметры, локальная область видимости в исходном коде определяется за счет использования правила именования данных идентификаторов.

# 1.18. Стандартная библиотека и её состав

В GRI-2018 присутствует стандартная библиотека includeGRI. Подключение стандартной библиотеки обязательно перед использованием функций. Библиотека представлена в таблице 1.8.

Таблица 1.8, Стандартная библиотека includeGRI

|  |  |
| --- | --- |
| Функция | Описание |
| int strlen (str идентификатор); | Находит длину строки. Применима только для идентификаторов типа переменная, тип данных str. |
| int double (int идентификатор); | Возведение числа в квадрат. Применима только для идентификаторов типа переменная, тип данных int. |
| str substr (str идентификатор, int идентификатор, int идентификатор); | Получение подстроки. Применима только для идентификаторов типа переменная, тип данных str и int. |

# 1.19. Ввод и вывод данных

Ввод данных в языке GRI-2018 не предусмотрен.

Вывод происходит с помощью оператора out.

Например: out(идентификатор), out("текст");

# 1.20. Точка входа

В языке GRI-2018 точкой входа является функция glavn. Наличие данной функции обязательно. В программе должна быть только одна точка входа. Имена идентификаторов не должны совпадать с именем точки входа.

# 1.21. Препроцессор

При трансляции языка GRI-2018 использование препроцессора в данной программе не предусмотрено.

# 1.22. Соглашения о вызовах

Отсутствуют.

# 1.23. Объектный код

Язык программирования GRI-2018 транслируется на Python.

# 1.24. Классификация сообщений транслятора

Транслятор, в ходе своей работы, генерирует сообщения, которые информируют пользователя о допущенных ошибках. Все сообщения транслятора разделены на интервалы, в зависимости от того на каком этапе была обнаружена ошибка. Все интервалы представлены в таблице 1.9.

Таблица 1.9, Сообщения транслятора

|  |  |
| --- | --- |
| Ошибки | Описание |
| 0,1,100,104,  110,112 | Ошибки системы |
| 102 | Ошибки входного кода |
| 5,105,111,  117,124,125 | Ошибки на этапе лексического анализа |
| 3,4,6,7,101,  106-109  113-116,  118-123,  125-127 | Ошибки на этапе семантического анализа |
| 8,600-605 | Ошибки на этапе синтаксического анализа |

# 1.25. Контрольный пример

Контрольный пример, демонстрирующий все возможности языка, GRI-2018 представлен в приложении И.

# 

# 2. Структура транслятора

# 2.1. Компоненты транслятора их назначение и принципы взаимодействия

Транслятор языка GRI-2018 разделён на отдельные части, которые взаимодействуют между собой и выполняют отведённые им функции. Алгоритм выполнения и описание каждой части приложения представлено в таблице 2.1.

Таблица 2.1, Структура транслятора

|  |  |
| --- | --- |
| Наименование  подпрограммы | Предназначение |
| Лексический анализатор | Обрабатывает входной файл исходного кода, проверяя его на разрешённые, запрещённые и игнорируемые символы. Преобразуя исходный код в более простой, с помощью замены длинных слов на лексемы, состоящие из одного символа, что упрощает последующую работу с кодом. Каждая лексема несёт в себе многочисленную информацию: имена идентификаторов, тип данных, тип переменной и так далее. После выполнения данной подпрограммы на выходе мы получаем таблицу лексем и таблицу идентификаторов. |
| Синтаксический анализатор | С помощью синтаксического анализатора проверяется правильность написанных конструкций по заданной грамматике. |
| Семантический анализатор | Реализован как отдельная часть. Нужен для того, чтобы проверить все правила, которые невозможно проверить на этапах лексического и синтаксического анализаторов. |

Продолжение таблицы 2.1

|  |  |
| --- | --- |
| Транслятор кода | Связан с работой лексического анализатора. Посредством полученных данных: таблицы лексем и идентификаторов, код на языке GRI-2018 транслируется в Python. |

# 2.2. Перечень параметров транслятора входных

Для указания запуска параметров запуска транслятора следует указывать параметры командной строки, которые представлены и описаны в таблице 2.2.

Таблица 2.2, Параметры транслятора

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Правило указания** | **Описание параметра** |
| -in: | -in:[полный путь к файлу] | Предназначен для определения местонахождения файла с исходным кодом, для проверки и разбора его на лексическом анализаторе.  Указание данного параметра обязательно. |
| -log: | -log:[полный путь к файлу] | Устанавливается местонахождение и имя файла, содержащего информацию о ходе работы транслятора.  Указание данного параметра необязательно. |

# 2.3. Перечень протоколов формируемых транслятором и их содержимое

Содержание лог файла может быть различно в зависимости от параметров командной строки, которые представлены в таблице 2.2*.* Наличие, а также описание информации, выводимой в лог-файл можно посмотреть в таблице2.3.

Таблица 2.3, Протоколы транслятора

|  |  |  |
| --- | --- | --- |
| **Тип информации** | **Описание информации** | **Наличие в файле протокола** |
| Дата и время трансляции | Выводит дату и время в формате: дд.мм. гг., чч: мм: сс. | Обязательно |
| Параметры командой строки | Выводит информацию об указанных параметрах командной строки. | Обязательно |
| Полная таблица лексем | Выводит таблицу лексем с информацией к каждой лексеме в log.txt. | Необязательно |
| Упрощённая таблица лексем | Выводит только лексемы без дополнительной информации. | Необязательно |
| Таблица идентификаторов | Выводит таблицу идентификаторов с дополнительной информацией. | Необязательно |
| Трассировочная информация синтаксического анализа | Выводит полную информацию о разборе таблицы лексем на синтаксическом анализаторе. | Необязательно |
| Правила разбора | Выводит правила по которым осуществился разбор исходного кода. | Необязательно |
| Польская нотация | Выводит польскую нотацию к конечной версии таблицы лексем. | Необязательно |

# 

# 3. Разработка лексического анализатора

# 3.1. Структура лексического анализатора

Входными данными для лексического анализатора является массив цепочек, предварительно сформированный на первичной обработке исходного кода программы. Выходными данными являются таблица лексем, таблица идентификаторов, а также протокол работы. Структура лексического анализатора GRI-2018 представлена на рисунке 2.



Рисунок 2 Структура лексического анализатора GRI-2018

# **3.2. Контроль входных символов**

Таблица входных символов представлена на рисунке 1. Реализация таблицы на языке C++ представлена в приложении А. Обозначения символов в таблице представлены в таблице 3.1.

Таблица 3.1, Контроль входных символов

|  |  |
| --- | --- |
| **Символы** | **Значение в таблице символов** |
| () \* + - = , / \ % | SEP |
| Двойные кавычки() | QUOTE |
| Запрещённый | F |
| Разрешённый | T |
| Игнорируемый | I |

# 3.3. Удаление избыточных символов

При считывании из файла с исходным кодом, в случае если считанный символ оказался пробелом или сепаратором, тогда мы проверяем предыдущий символ, если предыдущий символ являлся пробелом, значит, текущий пробел мы просто игнорируем и не заносим в массив.

Если встречается двойная кавычка, то ставится флаг, который не позволяет удалять избыточные символы. Повторная встреча кавычек снимает флаг.

# 3.4. Перечень ключевых слов, сепараторов, символов операций, соответствующих им лексем и конечных автоматов

Ключевые слова GRI-2018: value, int, str, function, back, strlen, substr, double, out, glavn. Сепараторы GRI-2018 представлены в таблице 1.1. Некоторые графы переходов представлены в приложении Б. Для каждой фразы также соответствует автомат (рисунок 3), по которому происходит разбор выражения. Автомат является – детерминированным, то есть имеет количество состояний.

S0 S1

o

u

t

Рисунок 3, Пример автомата для out

Проверка происходит следующим образом: на каждый автомат в массиве подаётся фраза и с помощью графа переходов происходит разбор, если разбор выполнен, то происходит заполнение таблицы лексем и при необходимости таблицы идентификаторов.

Каждой фразе соответствует отдельный символ, называемый лексемой. Соответствие фраз с такими лексемами представлено в таблице 3.2.

Таблица 3.2, Лексемы

|  |  |  |
| --- | --- | --- |
| **Фрагмент** | **Лексема** | **Примечание** |
| int  str | t | Тип данных |
| идентификатор | i | (Переменная, функция или параметр) |
| литералы | l | Данные |
| function | f | функция |
| back | b | Возврат значения из функции |
| out | o | Вывод |
| substr | r | Получение подстроки. |
| strlen | n | Нахождение длины строки |
| glavn | g | Главная функция |
| ; | ; | Сепаратор |
| { | { | Начало тела внешней функции |
| } | } | Конец тела внешней функции |
| ( | ( | Начало перечисления параметров функции |
| ) | ) | Конец перечисления параметров функции |
| = | = | Оператор присваивания |
| +  -  \*  / | +  -  \*  / | Арифметические операторы |
| , | , | Сепаратор параметров |
| includeGRI | B | Встроенная библиотека |
| value | d | Создание переменной |
| double | p | Возведение в квадрат |

# 3.5. Основные структуры данных

Код на языке С++ с описанием структур таблиц лексем и идентификаторов, приведён в приложении В.

# 3.6. Принцип обработки ошибок и их перечень

В случае если в результате работы лексического анализатора найдена ошибка, то анализатор заполняет структуру, содержащую ошибки, произошедшие в процессе лексического анализа. После чего вызывается функция генерации ошибки, в которую передается, в зависимости от места возникновения ошибки, следующая информация: код ошибки, номер строки в коде, номер позиции в строке или только код ошибки.

# 3.7. Структура и перечень сообщений лексического анализатора

Для обработки ошибок лексический анализатор использует таблицу с сообщениями. Структура сообщений содержит информацию о номере сообщения, номер строки и позицию, где было вызвано сообщение в исходном коде, информацию об ошибке. При возникновении сообщения, лексический анализатор продолжает работу с исходным кодом, проверяет его до конца и выводит оставшиеся сообщения об ошибках, если они существуют.

Перечень сообщений представлен в таблице 3.3.

Таблица 3.3, Сообщения лексического анализатора

|  |  |  |
| --- | --- | --- |
| Код | Сообщение | Пояснение |
| 1 | Отсутствие главной функции | Не удалось найти точки входа в программу – **glavn** функции. |
| 2 | Попытка переопределения | Попытка изменить уже существующий идентификатор. |
| 3 | Отсутствует предварительное определение | Использование идентификатора без его предварительного объявления |
| 4 | Превышен размер таблицы лексем | Переполнение таблицы лексем |

# 3.8. Параметры лексического анализатора и режимы его работы

Для выполнения лексического анализа требуются две очереди. Одна очередь содержит токены, полученные на этапе проверки исходного кода на допустимость символов. Вторая очередь содержит номера строк, соответствующих каждому токену из первой очереди.

# 3.9. Алгоритм лексического анализатора

Изначально лексический анализатор пробует разобрать токен при помощи набора конечных автоматов. В случае если токен не разобран, запоминается номер строки, в которой находился этот токен, и впоследствии будет выведено сообщение об ошибке. Если токен разобран, лексический анализатор анализирует лексему, соответствующую данному слову, и выполняет действия, описанные для данной лексемы. Лексический анализатор продолжает работать пока не будет разобрано последнее слово.

# 3.10. Контрольный пример

В результате работы лексического анализатора мы получим таблицу лексем, таблицу идентификаторов. Текст таблицы идентификаторов представлен в приложении В.

Текст таблицы лексем:

1 B;

2 tfi(ti,ti)

3 {

4 dti;

5 i=i+i;

6 bi;

7 }

9 g

10 {

11 dti;

12 dti;

13 dti;

14 dti;

15 dti;

16 dti;

17 dti;

18 i=l;

19 i=l;

20 i=l;

21 i=e(i);

22 i=s(i,l,l);

23 i=i(l,i);

24 i=p(l);

25 o(l);

26 o(i);

27 o(i);

28 o(i);

29 o(i);

30 o(i);

31 bl;

32 }

# 

# 4. Разработка синтаксического анализатора

# 4.1. Структура синтаксического анализатора

Структура работы синтаксического анализатора и входные, выходные данные представлены на рисунке 4.



Рисунок 4, Синтаксический анализатор

# 4.2. Грамматика языка

Грамматика для синтаксического разбора языка GRI-2018 представлена в таблице 4.1. Для описания языка, разбираемого синтаксического анализатором, применяют грамматики типа 2 – контекстно-свободные грамматики.

Грамматика типа 2:  - контекстно-свободная грамматика, где T – множество терминалов, N – множество нетерминалов (S,N,F,E,W,M), P – правила перехода, S – стартовый символ. Описание терминалов представлено в таблице 3.2.

В контекстно-свободной грамматике правила имеют вид: , где , ,  - словарь грамматики , где Т-правило нашей грамматики.

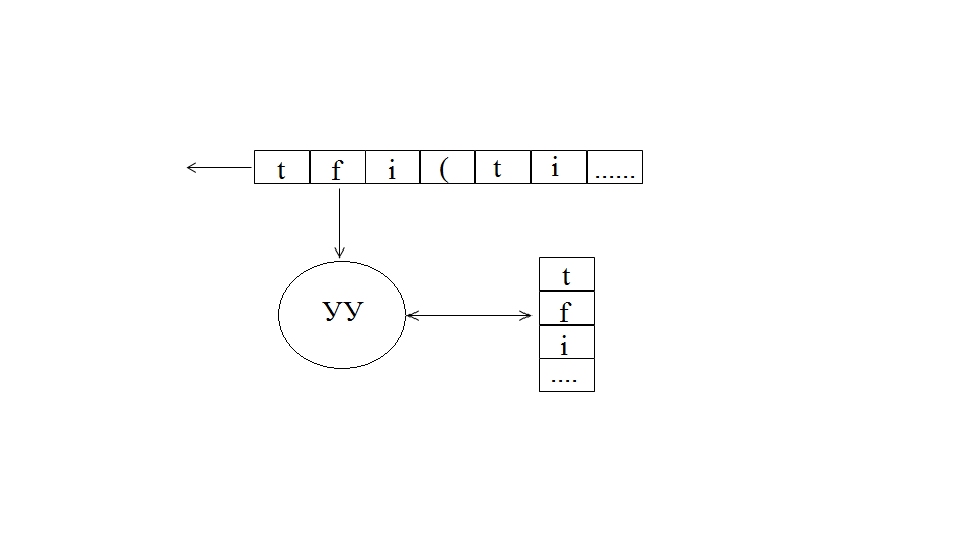
Таблица 4.1, Правила перехода нетерминальных символов

|  |  |
| --- | --- |
| Нетерминал | Правила |
| S – структура программы | B;S | tfi(F){N};S | t,f,i(F){N}S; |  t,f,i(F){N}S | t,f,i(){N}S | g{N}| |
| N – конструкции внутри функции | dti;N | dti=E;N | i=E;N | o=E;N | i=E;| bE; | dtp(F);N | dtfs(F);N | dte(F);N |
| F – параметры функции | ti | ti,F |
| E – выражения | i | l | iE | iM | lM | (W) | (E) | (E)M | s(W) | e(W) | p(W) | e(W)M | s(W)M| p(W)M| (W)M| i(W) | p(e(W)) | - l | -lME |
| W – параметры функции при вызове | l | i | i,W | l,W |
| M – знаки | \*E | /E |+E | -E | +EM | -EM |\*EM | /EM |

# 4.3. Построение конечного магазинного автомата

В данном курсовом проекте грамматика приведена к нормальной форме Грейбах. Это означает, что каждое правило имеет вид A → aα, где a∈T, α∈N.

Конечный автомат с магазинной памятью можно представить в виде семёрки , где М – автомат, Q – множество состояний, V – алфавит входных символов, Z – алфавит магазина,  - функция переходов,  – начальное состояние автомата,  – начальное состояние магазина, F – множество конечных состояний. Автомат с магазинной памятью представлен на рисунке 5.

Рисунок 5, Автомат с магазинной памятью.

Принцип работы автомата следующий: в магазин записывается стартовый символ;

* На основе полученных ранее таблиц формируется входная лента;
* Запускается автомат;
* Выбирается цепочка, соответствующая нетерминальному символу, записывается в магазин в обратном порядке;
* Если терминалы в стеке и в ленте совпадают, то данный терминал удаляется с ленты и стека. Иначе возвращаемся в предыдущее сохраненное состояние и выбираем другую цепочку нетерминала;
* Если в магазине встретился нетерминал, переходим к предыдущему пункту;
* Если наш символ достиг дна стека, и лента в этот момент пуста, то синтаксический анализ выполнен успешно. Иначе вызывается ошибка.

Структура состояний магазинного автомата представлена в виде фрагмента кода С++ (описание автомата находится в зоне комментариев) в приложении Г.

# 4.4. Основные структуры данных

Код С++ с описанием структур данных описывающих грамматику представлен в приложении Д.

# 4.5. Описание алгоритма синтаксического разбора

Алгоритм синтаксического разбора основан на работе автомата с магазинной памятью. Работа данного автомата описана в пункте 4.3.

# 4.6. Структура и перечень сообщений синтаксического анализатора

Перечень сообщений синтаксического анализатора представлен в таблице 4.2.

Таблица 4.2, Перечень сообщений синтаксического анализатора

|  |  |
| --- | --- |
| **Код ошибки** | **Сообщение** |
| 600 | Неверная структура программы. |
| 601 | Ошибка в операторах. |
| 602 | Ошибка в выражении. |
| 603 | Ошибка в параметрах функции. |
| 604 | Ошибка в конструкции функции. |
| 605 | Ошибка в принимаемых параметрах функции. |

# 

# 4.7. Параметры синтаксического анализатора и режимы его работы

Для управления результата работы синтаксического анализатора используются входные параметры, описанные в пункте 2.2Перечень входных параметров транслятора в таблице 2.2.

# 4.8. Принцип обработки ошибок

Изначально, если анализатор разбирает часть исходного кода по какому-либо правилу, то запоминает его. При возникновении ошибки синтаксический анализатор откатиться назад до правила, при помощи которого разбор был успешным, если это возможно. После чего пытается применить последующие правила из грамматики. В случае если правило невозможно подобрать выводится сообщение об ошибке.

# 4.9. Контрольный пример

Пример разбора исходного кода на языке программирования GRI-2018 синтаксическим анализатором представлен в виде дерева разбора в приложении Е.

# 

# 5. Разработка семантического анализатора

# 5.1. Структура семантического анализатора

Структура семантического анализатора представлена на рисунке 5.1.



Рисунок 5.1, Структура семантического анализатора

Семантический анализатор – часть транслятора, выполняющая семантический анализ, то есть исходный код проверяется на наличие ошибок, которые невозможно отследить при помощи регулярной и контекстно-свободной грамматики. Входными данными являются таблица лексем и идентификаторов.

# 5.2. Функции семантического анализатора

Функции семантического анализатора аналогичны ошибкам, приведённым в таблице 1.7. То есть каждой ошибке соответствует функция, которая проверяет наличие данной ошибки в исходном коде на языке GRI-2018.

# 5.3. Принцип обработки ошибок

В случае возникновения ошибок, вызываем функцию получения ошибки, которая принимает обязательным параметром код ошибки в таблице сообщений. Затем производится вывод ошибки в поток лог-файла, а также ошибка выводится в консоль.

# 

# 5.4. Контрольный пример

Контрольные примеры, предназначенные для проверки обработки некоторых возможных ошибок, представлены в виде текстовых файлов с исходным кодом на языке GRI-2018, которые прилагаются на переносном носителе CD-RW вместе с файлами решения под среду разработки MS Visual Studio 2017. Также несколько фрагментов кода с ошибкой, которые демонстрируют работу обработчика ошибок, представлены в приложении Ж.

# 6. Преобразование выражений

# 6.1. Выражения, допускаемые языком

В языке GRI-2018 допускаются вычисления выражений исключительно с целочисленными типами данных. Вычисление выражений в языке GRI-2018 происходит c преобразованием в польскую запись. Приоритет операций представлен в таблице 6.1.

Таблица 6.1, Приоритеты операций

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

# 6.2. Польская запись и принцип её построения

Польская запись - это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок. Существует два типа польской записи: прямая и обратная, также известные как префиксная и постфиксная. Отличие их от классического, инфиксного способа заключается в том, что знаки операций пишутся не между, а, соответственно, до или после аргументов.

Суть алгоритма заключается в следующем. Просматривается исходная строка символов S слева направо, операнды переписываются в выходную строку В, а знаки операций заносятся в стек, который первоначально пуст, на основе следующих правил:

* Если в строке S встретился операнд, то помещаем его в строку В;
* Если в S встретилась открывающая скобка, то помещаем ее в стек;
* Если в S встретилась закрывающая скобка, то выталкиваем из стека в строку В все операции до открывающей скобки, саму отрывающую скобку также извлекаем из стека; обе скобки (открывающая и закрывающая) игнорируются;
* Если в S встретилась операция Х, то выталкиваем из стека все операции, приоритет которых не ниже Х, после чего операцию Х записываем в стек;
* При достижении конца строки S, если стек не пуст, переписываем его элементы в выходную строку В.

# 6.3. Примеры преобразования выражений

Пример преобразования выражений к польской записи из контрольного примера.

value int result;

result = a+b;

В польской записи получаем

dti;

i=ii+;

# 

# 7. Генерация кода

Генерация кода – заключительный этап работы транслятора. Результатом данного этапа будет код, сгенерированный для выполнения Python на основе таблицы лексем и таблицы идентификаторов, что является требуемым результатом работы программы.Транслятор кода начинает свою работу только в том случае если код на языке GRI-2018 прошёл предыдущие компоненты транслятора без ошибок. Схема данного этапа изображена на рисунке 6.



Рисунок 6, Схема работы генератора кода

На вход подаются по очереди лексемы, и в зависимости от поданной лексемы языка GRI-2018, в файл Generation.py записывается соответствующее ей слово. После разбора последней лексемы будет получен файл с кодом на Python. После открытия этого файла, откроется командная строка и произойдёт выполнение программы. В таблице 7.1 представлены лексемы и соответствующие им слова для записи в файл.

Таблица 7.1, Трансляция на Python

|  |  |
| --- | --- |
| Лексема | Вывод |
| { | Переход на новую строку с табуляцией |
| d | Запись не производится |
| ; | Переход на новую строку |
| = | = |

Продолжение таблицы 7.1

|  |  |
| --- | --- |
| B | Import GRI as GRI |
| o | Print |
| ( | ( |
| ) | ) |
| } | Переход на новую строку с возвращением в начало строки |
| + | + |
| - | - |
| \* | \* |
| / | //(для округления) |
| f | def |
| , | , |
| b | return |
| e | len |
| p | GRI.Doub |
| s | GRI.Substr |

Весь сгенерированный код для контрольных примеров находится на CD-RW диске, приложенном к данному курсовому проекту.

# 

# 8. Тестирование транслятора

Тестирование транслятора GRI-2018 будет выполняться посредством компиляции исходного кода в проекте Visual Studio 2017 и запуска, полученного файла с исходным кодом на языке Python. Во время тестирования транслятора файл с исходным кодом GRI-2018 будет изменяться для проверки эффективности и работоспособности транслятора в различных ситуациях. Результат выполнения контрольного примера на рисунке 7.

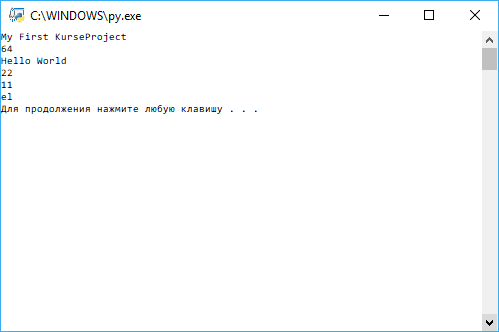


Рисунок 7, Результат выполнения на Python 3.6

Файлы решения под среду разработки MS Visual Studio 2017, а также файлы с исходным кодом на языке GRI-2018 прилагаются на переносном носителе CD-RW.

# 

# Заключение

В данном курсовом проекте были выполнены поставленные минимальные требования. В ходе работы было изучено много нового, а также закреплены знания, которые были получены ранее. При написании приложения были усвоены такие понятия как синтаксический, лексический и семантический анализатор и многие другие.

В итоге был получен примитивный язык программирования GRI–2018, который не имеет сложных конструкций, которые реализованы на сегодняшний день во многих других языках программирования.

Окончательная версия языка GRI -2018 включает:

1. 2 типа данных;
2. Поддержка оператора вывода;
3. Возможность подключения и вызова функций стандартной библиотеки, состоящей из четырех функций;
4. Наличие 4 арифметических операторов для вычисления выражений;
5. Возможность вызова функции в выражении.

# 

# Литература

1. Python [Электронный ресурс] ­– Режим доступа: <https://habr.com/hub/python/>- Дата доступа: 20.12.2018.

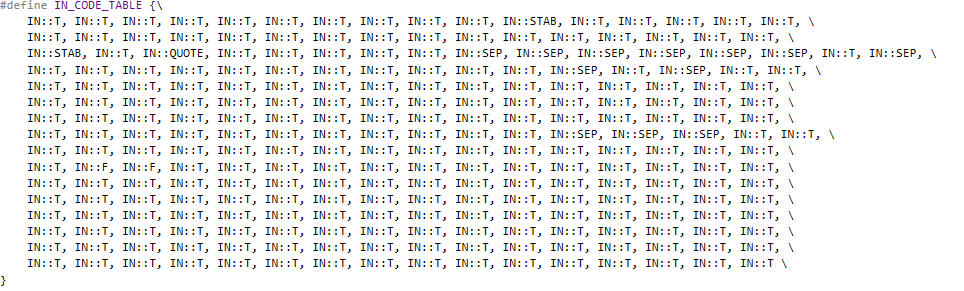
2. Наркевич А.С., Курс лекций по предмету языки программирования – 2018

3. Автомат с магазинной памятью [Электронный ресурс] – Режим доступа: <https://ru.wikipedia.org/wiki/Автомат_с_магазинной_памятью> - Дата доступа: 20.12.2018.

4. Windows 1251 [Электронный ресурс] – Режим доступа: <https://en.wikipedia.org/wiki/Windows-1251>- Дата доступа: 20.12.2018.

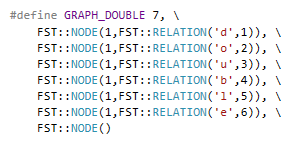
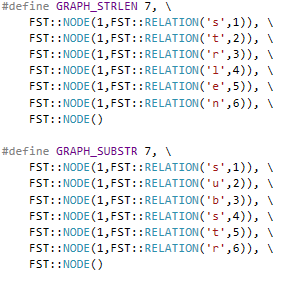
5. Обратная польская запись[Электронный ресурс] – Режим доступа: <https://ru.wikipedia.org/wiki/Обратная_польская_запись> - Дата доступа: 20.12.2018.

# Приложение А: Таблица входных символов.

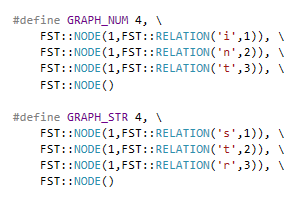


# Приложение Б: Графы переходов

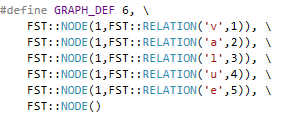
**Граф функций стандартной библиотеки**



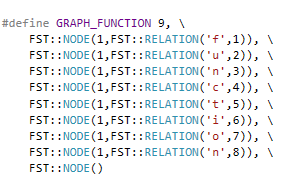
**Графы типов данных**



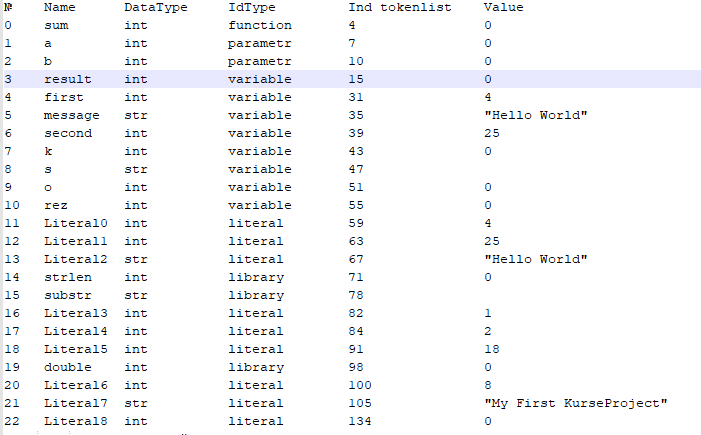
**Граф объявления**



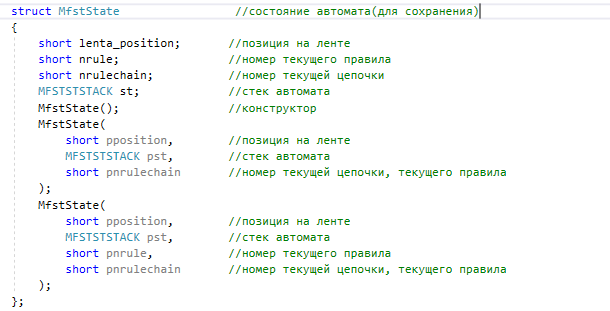
**Граф функции**



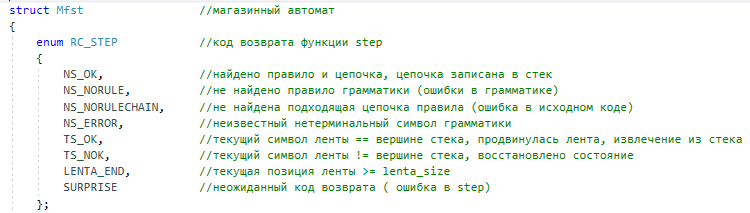
**Приложение В: Таблица идентификаторов**

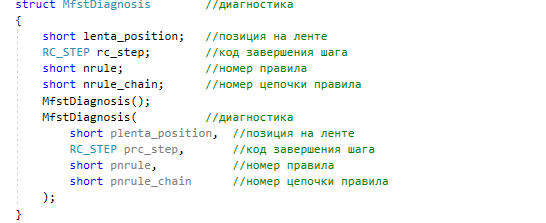


# Приложение Г: Состояния автомат с магазинной памятью



# Приложение Д: Автомат с магазинной памятью





# Приложение Е: Дерево разбора

0 : S->B;S

2 : S->tfi(F){N}S

6 : F->ti,F

9 : F->ti

13 : N->dti;N

17 : N->i=E;N

19 : E->iM

20 : M->+E

21 : E->i

23 : N->bE;

24 : E->i

27 : S->g{N}

29 : N->dti;N

33 : N->dti;N

37 : N->dti;N

41 : N->dti;N

45 : N->dti;N

49 : N->dti;N

53 : N->dti ;N

57 : N->i=E;N

59 : E->l

61 : N->i=E;N

63 : E->l

65 : N->i=E;N

67 : E->l

69 : N->i=E;N

71 : E->e(W)

73 : W->i

76 : N->i=E;N

78 : E->s(W)

80 : W->i,W

82 : W->l,W

84 : W->l

87 : N->i=E;N

89 : E->iE

90 : E->(W)

91 : W->l,W

93 : W->i

96 : N->i=E;N

98 : E->p(W)

100 : W->l

103 : N->o(E);N

105 : E->l

108 : N->o(E);N

110 : E->i

113 : N->o(E);N

115 : E->i

118 : N->o(E);N

120 : E->i

123 : N->o(E);N

125 : E->i

128 : N->o(E);N

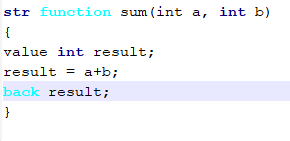
130 : E->i

133 : N->bE;

134 : E->l

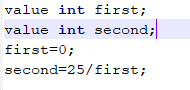
# **Приложение Ж**: Тестирование семантического анализатора

Пример:



Ошибка 127: [Семантическая] Тип функции и тип возвращаемого значения не совпадают, строка 2.

Пример:



Ошибка 109: [Семантическая] Обнаружено невозможное деление на ноль, строка 19.

# Приложение И: Контрольный пример.

