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 HDV-2019»

Выполнил студент Гунько Дмитрий Витальевич

(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.)

(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 2019

Содержание

[Глава 1. Спецификация языка программирования 6](#_Toc501592483)

[1.1 Характеристика языка программирования 6](#_Toc501592484)

[1.2 Алфавит языка 6](#_Toc501592485)

[1.3 Применяемые сепараторы 7](#_Toc501592486)

[1.4 Применяемые кодировки 7](#_Toc501592487)

[1.5 Типы данных 8](#_Toc501592488)

[1.6 Преобразование типов данных 8](#_Toc501592489)

[1.7 Идентификаторы 8](#_Toc501592490)

[1.8 Литералы 9](#_Toc501592491)

[1.9 Объявления данных и область видимости 9](#_Toc501592492)

[1.10 Инициализация данных 9](#_Toc501592493)

[1.11 Инструкции языка 10](#_Toc501592494)

[1.12 Операции языка 10](#_Toc501592495)

[1.13 Выражения и их вычисления 11](#_Toc501592496)

[1.14 Программные конструкции языка 11](#_Toc501592497)

[1.15 Область видимости идентификаторов 11](#_Toc501592498)

[1.16 Семантические проверки 11](#_Toc501592499)

[1.17 Распределение оперативной памяти на этапе выполнения 12](#_Toc501592500)

[1.18 Стандартная библиотека и её состав 12](#_Toc501592501)

[1.19 Ввод и вывод данных 12](#_Toc501592502)

[1.20 Точка входа 13](#_Toc501592503)

[1.21 Препроцессор 13](#_Toc501592504)

[1.22 Соглашения о вызовах 13](#_Toc501592505)

[1.23 Объектный код 13](#_Toc501592506)

[1.24 Классификация сообщений транслятора 13](#_Toc501592507)

[1.25 Контрольный пример 13](#_Toc501592508)

[Глава 2. Структура транслятора 14](#_Toc501592509)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 14](#_Toc501592510)

[2.2 Перечень входных параметров транслятора 14](#_Toc501592511)

[2.3 Перечень протоколов, формируемых транслятором и их содержимое 14](#_Toc501592512)

[Глава 3. Разработка лексического анализатора 14](#_Toc501592513)

[3.1 Структура лексического анализатора 14](#_Toc501592514)

[3.2 Контроль входных символов 14](#_Toc501592515)

[3.3 Удаление избыточных символов 14](#_Toc501592516)

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

[3.6 Принцип обработки ошибок 14](#_Toc501592518)

[3.7 Структура и перечень сообщений лексического анализатора 14](#_Toc501592519)

[Глава 4. Разработка синтаксического анализатора 15](#_Toc501592520)

[4.1 Структура синтаксического анализатора. 15](#_Toc501592521)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 15](#_Toc501592522)

[4.3 Построение конченого магазинного автомата 15](#_Toc501592523)

[4.4 Основные структуры данных 15](#_Toc501592524)

[4.5 Описание алгоритма синтаксического разбора 15](#_Toc501592525)

[4.6 Структура и перечень сообщений синтаксического анализатора 15](#_Toc501592526)

[4.7 Параметры синтаксического анализатора и режимы его работы 15](#_Toc501592527)

[4.8 Принцип обработки ошибок 15](#_Toc501592528)

[4.9 Контрольный пример 15](#_Toc501592529)

[Глава 5. Разработка семантического анализатора 15](#_Toc501592530)

[5.1 Структура семантического анализатора 15](#_Toc501592532)

[5.2 Функции семантического анализатора 15](#_Toc501592533)

[5.3 Структура и перечень сообщений семантического анализатора 15](#_Toc501592534)

[5.4 Принцип обработки ошибок 15](#_Toc501592535)

[5.5 Контрольный пример 15](#_Toc501592536)

[Глава 6. Преобразование выражений 15](#_Toc501592537)

[6.1 Выражения, допускаемые языком 15](#_Toc501592538)

[6.2 Польская запись и принцип ее построения 15](#_Toc501592539)

[6.3 Программная реализация обработки выражений 15](#_Toc501592540)

[6.4 Контрольный пример 15](#_Toc501592541)

[Глава 7. Генерация кода 16](#_Toc501592542)

[7.1 Структура генератора кода 16](#_Toc501592543)

[7.2 Представление типов данных в памяти 16](#_Toc501592544)

[7.3 Статическая библиотека 16](#_Toc501592545)

[7.4 Особенности алгоритма генерации кода 16](#_Toc501592546)

[7.5 Контрольный пример 16](#_Toc501592547)

[Глава 8. Тестирование транслятора 16](#_Toc501592548)

[Заключение 16](#_Toc501592549)

[Приложение А 16](#_Toc501592550)

[Приложение Б 16](#_Toc501592551)

[Приложение Г 16](#_Toc501592552)

[Приложение Д 16](#_Toc501592553)

[Приложение Е 16](#_Toc501592554)

[Литература 16](#_Toc501592555)

**Введение**

Императивный язык программирования HDV-2019 является процедурным, строго типизированным, транслируемым, со своим байт-кодом.

Императивное программирование — это [парадигма программирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) (стиль написания [исходного кода](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) [компьютерной программы](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0)), для которой характерно следующее:

* в исходном коде программы записываются [инструкции](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) (команды);
* инструкции должны выполняться последовательно;
* данные, получаемые при выполнении предыдущих инструкций, могут читаться из памяти последующими инструкциями;
* данные, полученные при выполнении инструкции, могут записываться в память.

Процедурный язык программирования — язык высокого уровня, в котором используется метод разбиения программ на отдельные связанные между собой модули — подпрограммы (процедуры, блоки и функции).

Байт — стандартное [промежуточное представление](https://ru.wikipedia.org/w/index.php?title=%D0%9F%D1%80%D0%BE%D0%BC%D0%B5%D0%B6%D1%83%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%B5%D0%B4%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5&action=edit&redlink=1), в которое может быть переведена [компьютерная программа](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) автоматическими средствами. По сравнению с [исходным кодом](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4), удобным для создания и чтения человеком, байт-код — это компактное представление программы, уже прошедшей [синтаксический](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7) и семантический анализ. В нём в явном виде закодированы [типы](https://ru.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), [области видимости](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BB%D0%B0%D1%81%D1%82%D1%8C_%D0%B2%D0%B8%D0%B4%D0%B8%D0%BC%D0%BE%D1%81%D1%82%D0%B8) и другие конструкции. С технической точки зрения байт-код представляет собой машинно-независимый [код низкого уровня](https://ru.wikipedia.org/wiki/%D0%9D%D0%B8%D0%B7%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), генерируемый [транслятором](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D1%81%D0%BB%D1%8F%D1%82%D0%BE%D1%80) из исходного кода.

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

Классический транслятор состоит из следующих частей:

* лексический анализатор;
* синтаксический анализатор;
* семантический анализатор;
* генератор кода, или интерпретатор.

Все части транслятора, взаимодействуя между собой, обрабатывают входной текст и строят для него эквивалентный текст на байт-коде.

# Глава 1. Спецификация языка программирования

* 1. **Характеристика языка программирования**

Императивный язык программирования HDV-2019 является процедурным, строго типизированным, транслируемым.

* 1. **Алфавит языка**

Алфавит HDV-2019 основан на таблице символов ACSII представленной на рисунке 1.1



Рисунок 1.1 - Алфавит входных символов

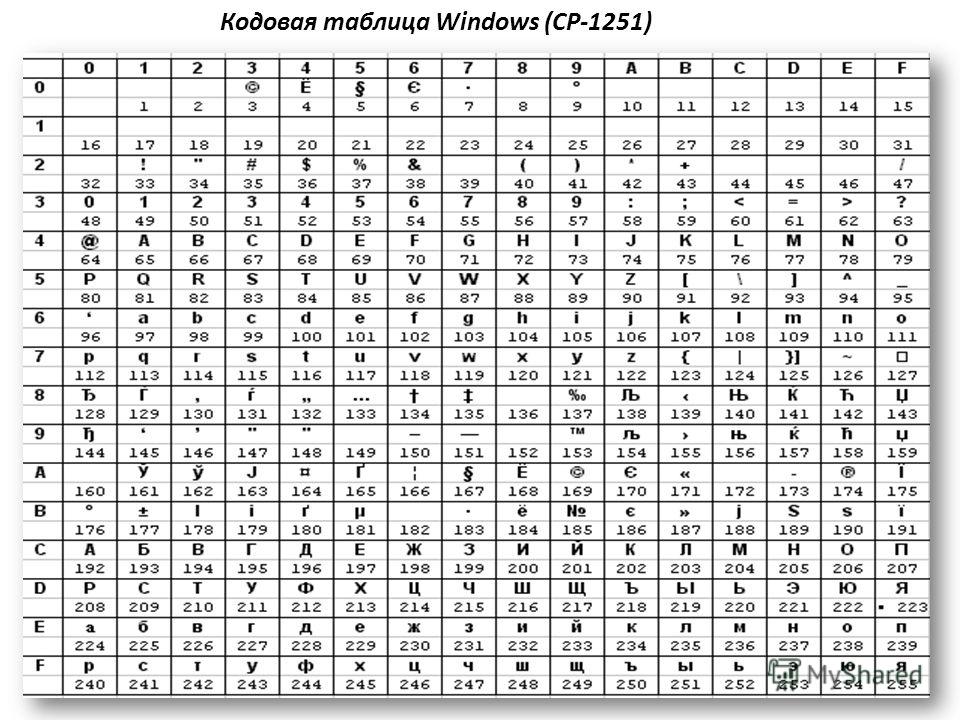
* 1. **Применяемые сепараторы**

Сепараторы служат для разделения операций языка. Сепараторы, применяемые в языке программирования HDV-2019, приведены в таблице 1.1.

Таблица 1.1 - Применяемые сепараторы

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

* 1. **Применяемые кодировки**

При трансляции применяется кодировка Windows – 1251. Описание кодировки представлено на рисунке 1.2. 

* 1. **Типы данных**

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

Таблица 1.2 - Фундаментальные типы данных

|  |  |
| --- | --- |
| Тип данных | Описание |
| int | Является знаковым целочисленным типом данных (4 байта). Предназначен для арифметических операций над числами. Инициализация по умолчанию: 0.  Предусмотрены следующие арифметические операции:  + - бинарная операция суммирования (число + число);  - - бинарная операция вычитания (число - число);  \* - бинарная операция умножения (число \* число);  / - бинарная операция деления (число / число);  \*\* - бинарная операция возведения в степень (число \*\* число). |
| u\_int | Является беззнаковым целочисленным типом данных (4 байта). Предназначен для арифметических операций над числами. Инициализация по умолчанию: 0.  Предусмотрены следующие арифметические операции:  + - бинарная операция суммирования (число + число);  - - бинарная операция вычитания (число - число);  \* - бинарная операция умножения (число \* число);  / - бинарная операция деления (число / число);  \*\* - бинарная операция возведения в степень (число \*\* число). |
| str | Является строковым типом данных. Используется для работы с символами, каждый символ занимает 1 байт. Максимальное количество символов – 255.  Инициализация по умолчанию: нуль-символ.  Операции над данными строкового типа: возможно присваивание строковому идентификатору значения другого строкового идентификатора, строкового литерала или значения строковой функции, конкатенация строк. |
| boolean | Является логическим типом данных (1 байта). Предназначен для логических операций(И, ИЛИ, НЕ). Инициализация по умолчанию: false. |

* 1. **Преобразование типов данных**

Преобразование среди типов данных поддерживается.

* 1. **Идентификаторы**

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

Имя идентификатора составляется по следующим правилам:

* состоит из символов английского алфавита [a..z] и ‘\_’.
* длина идентификатора не должна превышать 25 символов. При превышении максимально допустимой длины длина идентификатора усекается до 25.
  1. **Литералы**

Предусмотрены числовые и строковые литералы. Правила записи приведены в таблице 1.3.

Таблица 1.3 - Правила записи литералов

|  |  |
| --- | --- |
| Тип литерала | Описание литерала |
| Числовые | Обязательно содержат в себе только цифры [0..9] без дробной части. Максимально допустимым значением является число . Минимально допустимым значением является . Предусмотрена запись литералов в двоичной и восьмеричной формах. При выходе за пределы допустимости, выводится соответствующая ошибка. |
| Строковые | Символы, заключенные в “…” (двойные кавычки), число которых не превышает 255. Доступны все символы знаки препинания, символы английского и русского алфавитов. При выходе за пределы допустимости, выводится соответствующая ошибка. |

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

В языке программирования HDV\_2019 необходимо объявить переменную до ее использования. Областью видимости переменной является блок, в которой она определена (тело цикла, функции, условного оператора или блока кода). Объявление глобальных переменных осуществляется с помощью ключевого слова global. Область видимости построена по принципу C++ (сверху вниз). Для объявления переменных используется следующая конструкция:

[global] ˽ <тип данных> ˽ <идентификатор>;

* 1. **Инициализация данных**

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

* 1. **Инструкции языка**

В языке программирования HDV-2019 применяются инструкции, представленные в таблице 1.4.

Таблица 1.4 - Инструкции языка

|  |  |
| --- | --- |
| Инструкция языка | Синтаксис |
| Объявление переменной | <тип данных>˽<идентификатор>; |
| Объявление глобальной переменной | global ˽ <тип данных>˽<идентификатор>; |
| Объявление функции | <тип данных>˽func˽<идентификатор>([параметры])  {тело функции}; |
| Присваивание | str:  <идентификатор> = <литерал>;  <идентификатор> = <выражение>;  <идентификатор 1> = <идентификатор 2>;  int и u\_int:  <идентификатор> = <литерал>;  <идентификатор> = <выражение>;  <идентификатор 1> = <идентификатор 2>;  boolean:  <идентификатор> = <литерал>;  <идентификатор> = <выражение>;  <идентификатор 1> = <идентификатор 2>; |
| Главная функция | main  {  <<инструкции языка>  } |
| Вызов функций | <идентификатор функции> ([параметры]); |
| Вывод данных в консоль | cprint <идентификатор, литерал>; |
| Возврат из функции | return <идентификатор>;  return <литерал>; |

* 1. **Операции языка**

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

Таблица 1.5 - Операции языка

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

* 1. **Выражения и их вычисления**

Предусмотрены следующие правила составления выражений:

* + выражения записываются до ввода сепаратора “;”;
  + рассматриваются слева направо;
  + для изменения приоритета операция используются круглые скобки ();
  + реализация выражений происходит с помощью обратной польской записи.
  1. **Программные конструкции языка**

Программные конструкции представлены в таблице 1.7.

Таблица 1.7 - Программные конструкции языка

|  |  |
| --- | --- |
| Конструкция | Представление в языке |
| Главная функция | main  {  <<инструкции языка>  } |
| Функция | <тип данных> ˽ func ˽ <идентификатор>([параметры])  {  <инструкции языка>  } |
| Точка входа | main |

* 1. **Область видимости идентификаторов**

Все идентификатор, объявленные внутри функции, являются локальными. Глобальных переменные, объявленные вне какого-либо блока видны во всей программе. Параметры видны только внутри функции, в которой объявлены.

* 1. **Семантические проверки**

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

Таблица 1.8 - Перечень семантических проверок

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

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

Транслированный код использует две области памяти. В сегмент констант заносятся все литералы. В сегмент данных заносятся переменные и параметры функций. Локальная область видимости в исходном коде определяется за счет использования переменной, хранящей имя родительского блока, что и обуславливает их локальность на уровне исходного кода, все глобальные переменные имеют имя родительского блока “global”.

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

В языке HDV-2019 присутствует стандартная библиотека, которая подключается автоматически при трансляции исходного кода в байт-код. Содержимое библиотеки и описание функций представлено в таблице 1.9.

Таблица 1.9 - Стандартная библиотека языка KYA-2018

|  |  |
| --- | --- |
| Функция | Описание |
| str func getDate(); | Строковая функция. Вычисляет и возвращает текущую дату в формате dd.mm.yy; |
| str func getTime(); | Строковая функция. Вычисляет и возвращает текущее время в формате hh:mm::ss; |

* 1. **Ввод и вывод данных**

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

cprint <выражение>;

* 1. **Точка входа**

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

* 1. **Препроцессор**

Препроцессор в языке HDV-2019 не предусмотрен.

* 1. **Соглашения о вызовах**

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

* 1. **Объектный код**

Язык транслируется в байт-код.

* 1. **Классификация сообщений транслятора**

Классификация осуществлена по номеру ошибки.

Серии ошибок:

* 0-99 - системные ошибки
* 100-109 - ошибки параметров
* 110-119 - ошибки открытия и чтения файлов
* 120-129 - лексические ошибки
  1. **Контрольный пример**

Контрольный пример, написанный на языке HDV-2019, представлен в приложении А.

**Глава 2. Структура транслятора**

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

**2.2 Перечень входных параметров транслятора**

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

# Глава 3. Разработка лексического анализатора

**3.1 Структура лексического анализатора**

**3.2 Контроль входных символов**

**3.3 Удаление избыточных символов**

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

**3.5 Основные структуры данных**

**3.6 Принцип обработки ошибок**

## 3.7 Структура и перечень сообщений лексического анализатора

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

**3.9 Контрольный пример**

# Глава 4. Разработка синтаксического анализатора

## 4.1 Структура синтаксического анализатора.

## 4.2 Контекстно-свободная грамматика, описывающая синтаксис языка

## 4.3 Построение конченого магазинного автомата

## 4.4 Основные структуры данных

## 4.5 Описание алгоритма синтаксического разбора

## 4.6 Структура и перечень сообщений синтаксического анализатора

## 4.7 Параметры синтаксического анализатора и режимы его работы

## 4.8 Принцип обработки ошибок

## 4.9 Контрольный пример

# Глава 5. Разработка семантического анализатора

1. 1. **Структура семантического анализатора**
   2. **Функции семантического анализатора**
   3. **Структура и перечень сообщений семантического анализатора**
   4. **Принцип обработки ошибок**
   5. **Контрольный пример****Глава 6. Преобразование выражений**

**6.1 Выражения, допускаемые языком**

**6.2 Польская запись и принцип ее построения**

**6.3 Программная реализация обработки выражений**

**6.4 Контрольный пример**

# Глава 7. Генерация кода

## 7.1 Структура генератора кода

## 7.2 Представление типов данных в памяти7.3 Статическая библиотека

## 7.4 Особенности алгоритма генерации кода

## 7.5 Контрольный пример

# Глава 8. Тестирование транслятора

# Заключение

# Приложение А

# Приложение Б

**Приложение В**

# Приложение Г

# Приложение Д

# Приложение Е

# Литература