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 MAV-2020»

Выполнил студент Михалькевич Алексей Вячеславович

(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 Наталья Владимировна

(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 2020

[Введение 4](#_Toc532814711)

[Глава 1 Спецификация языка программирования 5](#_Toc532814712)

[**1.1 Характеристика языка программирования** 5](#_Toc532814713)

[**1.2 Алфавит языка** 5](#_Toc532814714)

[**1.3 Применяемые сепараторы** 6](#_Toc532814715)

[**1.4 Применяемые кодировки** 6](#_Toc532814716)

[**1.5 Типы данных** 6](#_Toc532814717)

[**1.6 Преобразование типов данных** 6](#_Toc532814718)

[**1.7 Идентификаторы** 7](#_Toc532814719)

[**1.8 Литералы** 7](#_Toc532814720)

[**1.9 Объявление данных и область видимости** 8](#_Toc532814721)

[**1.10 Инициализация данных** 8](#_Toc532814722)

[**1.11 Инструкции языка** 8](#_Toc532814723)

[**1.12 Операции языка** 9](#_Toc532814724)

[**1.13 Выражения и их вычисления** 9](#_Toc532814725)

[**1.14 Программные конструкции языка** 10](#_Toc532814726)

[**1.15 Область видимости идентификаторов** 10](#_Toc532814727)

[**1.16 Семантические проверки** 10](#_Toc532814728)

[**1.17 Распределение оперативной памяти на этапе выполнения** 11](#_Toc532814729)

[**1.18 Стандартная библиотека и ее состав** 11](#_Toc532814730)

[**1.19 Ввод и вывод данных** 12](#_Toc532814731)

[**1.20 Точка входа** 12](#_Toc532814732)

[**1.21 Препроцессор** 12](#_Toc532814733)

[**1.22 Соглашения о вызовах** 12](#_Toc532814734)

[**1.23 Объектный код** 12](#_Toc532814735)

[**1.24 Классификация сообщений транслятора** 12](#_Toc532814736)

[**1.25 Контрольный пример** 13](#_Toc532814737)

[Глава 2 Структура транслятора 14](#_Toc532814738)

[**2.1 Компоненты транслятора их назначение и принципы взаимодействия** 14](#_Toc532814739)

[**2.2 Перечень входных параметров транслятора** 15](#_Toc532814740)

[**2.3 Перечень протоколов, формируемых транслятором и их содержимое** 15](#_Toc532814741)

[Глава 3 Разработка лексического анализатора 17](#_Toc532814742)

[**3.1 Структура лексического анализатора** 17](#_Toc532814743)

[**3.2 Контроль входных символов** 17](#_Toc532814744)

[**3.3 Удаление избыточных символов** 18](#_Toc532814745)

[**3.4 Перечень ключевых слов, сепараторов, символов операций соответствующим им лексемам и конечных автоматов** 18](#_Toc532814746)

[**3.5 Основные структуры данных** 19](#_Toc532814747)

[**3.6 Принцип обработки ошибок** 20](#_Toc532814748)

[**3.7 Структура и перечень сообщений лексического анализатора** 20](#_Toc532814749)

[**3.8 Параметры лексического анализатора и режимы его работы** 20](#_Toc532814750)

[**3.9 Алгоритм лексического анализа** 20](#_Toc532814751)

[**3.10 Контрольный пример** 21](#_Toc532814752)

[Глава 4 Разработка синтаксического анализатора 22](#_Toc532814753)

[**4.1 Структура синтаксического анализатора** 22](#_Toc532814754)

[**4.2 Контекстно-свободная грамматика, описывающая синтаксис языка** 22](#_Toc532814755)

[**4.3 Построение конечного магазинного автомата** 23](#_Toc532814756)

[**4.4 Основные структуры данных** 24](#_Toc532814757)

[**4.5 Описание алгоритма синтаксического разбора** 24](#_Toc532814758)

[**4.6 Структура и перечень сообщений синтаксического анализатора** 24](#_Toc532814759)

[**4.7 Параметры синтаксического анализатора и режимы его работы** 25](#_Toc532814760)

[**4.8 Принцип обработки ошибок** 25](#_Toc532814761)

[**4.9 Контрольный пример** 25](#_Toc532814762)

[Глава 5 Разработка семантического анализатора 26](#_Toc532814763)

[**5.1 Структура семантического анализатора** 26](#_Toc532814764)

[**5.2 Функции семантического анализа** 26](#_Toc532814765)

[**5.3 Структура и перечень сообщений семантического анализатора** 26](#_Toc532814766)

[**5.4 Принцип обработки ошибок** 27](#_Toc532814767)

[**5.5 Контрольный пример** 27](#_Toc532814768)

[Глава 6 Вычисление выражений 28](#_Toc532814769)

[**6.1 Выражения, допускаемые языком** 28](#_Toc532814770)

[**6.2 Польская запись и принцип ее построения** 28](#_Toc532814771)

[**6.3 Программная реализация обработки выражений** 29](#_Toc532814772)

[**6.4 Контрольный пример** 29](#_Toc532814773)

[**Глава 7 Генерация кода** 30](#_Toc532814774)

[**7.1 Структура генератора кода** 30](#_Toc532814775)

[**7.2 Представление типов данных в оперативной памяти** 30](#_Toc532814776)

[**7.3 Статическая библиотека** 31](#_Toc532814777)

[**7.4 Особенности алгоритма генерации кода** 31](#_Toc532814778)

[**7.5 Входные параметры генератора кода** 31](#_Toc532814779)

[**7.6 Контрольный пример** 31](#_Toc532814780)

[Глава 7 Генерация кода 30](#_Toc532814774)

[**7.1 Структура генератора кода** 30](#_Toc532814775)

[**7.2 Представление типов данных в оперативной памяти** 30](#_Toc532814776)

[**7.3 Статическая библиотека** 31](#_Toc532814777)

[**7.4 Особенности алгоритма генерации кода** 31](#_Toc532814778)

[**7.5 Входные параметры генератора кода** 31](#_Toc532814779)

[**7.6 Контрольный пример** 31](#_Toc532814780)

[Глава 8 Тестирование транслятора 32](#_Toc532814781)

[**8.1 Общие положения** 32](#_Toc532814782)

[**8.2 Результаты тестирования** 32](#_Toc532814783)

[Заключение 33](#_Toc532814784)

[Список использованных источников 34](#_Toc532814785)

[Приложение А 35](#_Toc532814786)

[Приложение Б 36](#_Toc532814787)

[Приложение В 37](#_Toc532814788)

[Приложение Г 41](#_Toc532814789)

# **Введение**

Задачей данного курсового проекта является разработка компилятора для языка программирования MAV-2020.

Компилятор — это частный случай (разновидность) трансляторов. А транслятор — это программа, принимающая на вход программу на одном языке (в нашем случае языке MAV-2020), и преобразующая её в программу на другом языке (в нашем случае транслируя в MASM32 под windows)

Интерпретация — построчный анализ, обработка и выполнение исходного кода программы или запроса.

Исходя из цели курсового проекта, были определены следующие задачи:

* разработать спецификацию языка программирования;
* разработать структуру транслятора;
* разработать лексический, синтаксический и семантический анализаторы;
* преобразовать выражения;
* произвести генерацию кода, т.е. транслировать язык программирования MAV-2020 в MASM32;
* произвести тестирование.

Решения каждой из поставленных задач будут приведены в соответствующих главах курсового проекта.

В первой главе работы определена спецификация языка программирования, т.е. описан синтаксис и семантика языка.

Во второй главе работы представлена структура транслятора, т.е. перечислены компоненты транслятора, их назначение и принципы взаимодействия, перечень входных параметров, перечень протоколов, формируемых транслятором и их содержимое.

В третьей главе работы показана разработка лексического анализатора, порождающего таблицы лексем и идентификаторов.

# **Глава 1 Спецификация языка программирования**

## **Характеристика языка программирования**

Язык программирования MAV-2020 предназначен для выполнения простейших арифметических действий и операций над строками.

Язык программирования MAV-2020 является строго типизированным, процедурным, компилируемым.

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

Строго типизированный язык программирования — язык, в котором переменные привязаны к конкретным типам данных. Язык не позволяет смешивать в выражениях различные типы и не выполняет автоматические неявные преобразования.

Компилируемый язык программирования — язык программирования, исходный код которого преобразуется компилятором в исходный код на другом языке программирования.

## **Алфавит языка**

Алфавит языка MAV-2020 основан на кодировке Windows-1251. Таблица кодировки Windows-1251 представлена на рисунке 1.1.

Исходный код может содержать символы латинского алфавита, цифры десятичной системы счисления от 0 до 9, восьмеричной 0 до 7

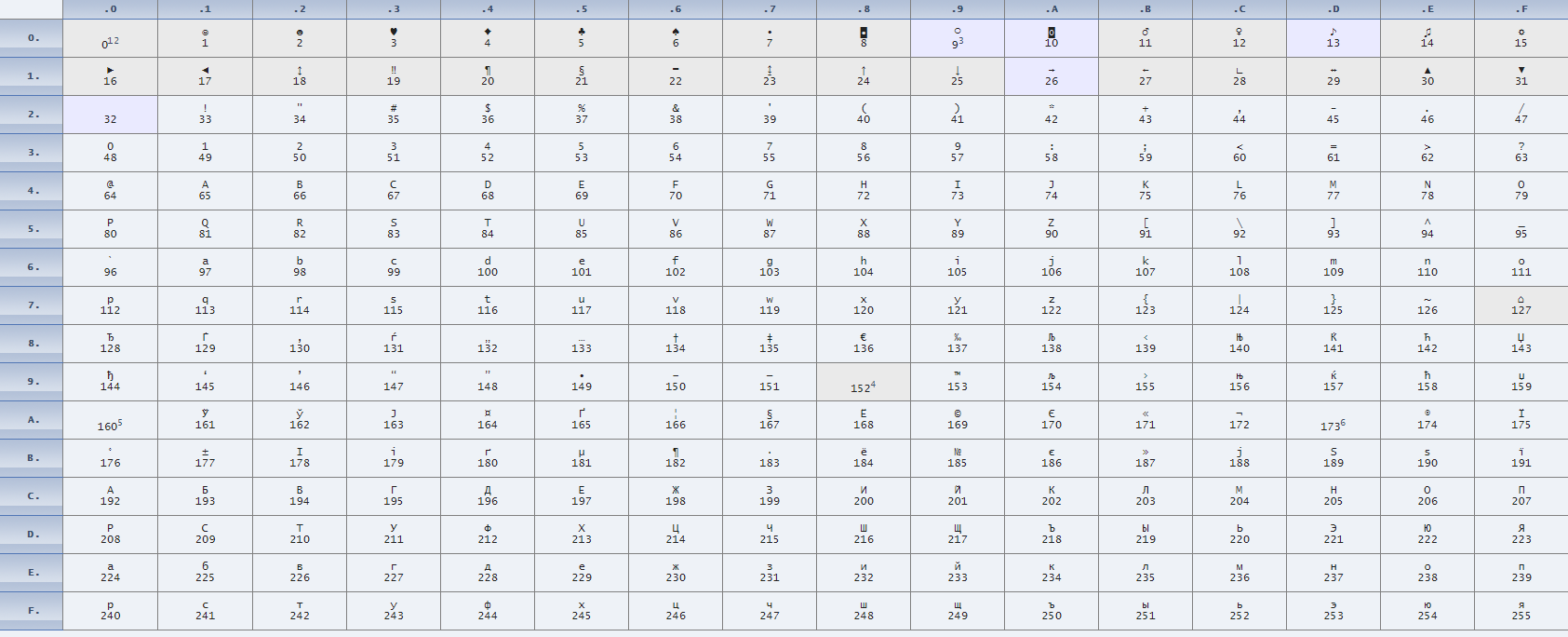


Рисунок 1.1 — Основная таблица Windows-1251

## **Применяемые сепараторы**

Применяемые сепараторы в языке MAV-2020, приведены в таблице 1.1. Таблица 1.1 — Применяемые сепараторы

|  |  |
| --- | --- |
| Сепаратор | Область применения |
| « » (пробел) | Допускается везде, кроме идентификаторов и ключевых слов |
| ; | Разделение конструкций |
| { } | Заключение программного блока |
| = | Присвоение значения |
| ( ) | Приоритет операций (в выражениях), параметры функции |
| , | Разделение параметров |
| +,-,\*,/, % | Арифметические операции |

## **Применяемые кодировки**

Для написания исходного кода на языке программирования MAV-2020 используется кодировка Windows-1251, которая представлена на рисунке 1.1.

## **Типы данных**

В язык MAV-2020 предусмотрены два типа данных: целочисленный и строковый, представленные в таблице 1.2.

Таблица 1.2 — Типы данных

| Тип | Описание |
| --- | --- |
| Int | Целочисленный тип данных. В памяти занимает 2 байта. Автоматическая инициализация 0 (нулем). Максимальное значение +32767, минимальное −32767. |
| String | Строковый тип данных. Один символ занимает 1 байт. Максимальная длина строки (число символов) составляет 254 символа. При объявлении автоматически инициализируется длинной 0, и нуль-терминантом строки “\0”. |
| Boolean | Булевый тип данных. В памяти занимает 1 байт. Автоматическая инициализация 0 (false). Может принимать значение 1 (true) | 0 (false). |

## **Преобразование типов данных**

Преобразование типов данных в языке MAV-2020 не поддерживается.

## **Идентификаторы**

Идентификатор — имя компонента программы (переменной или функции), составленное программистом по определенным правилам.

В идентификаторах языка MAV-2020 разрешается использовать латинские буквы верхнего и нижнего регистров.

Идентификаторы не могут совпадать с ключевыми словами. Данные правила справедливы как для переменных, так и для функций.

Примеры правильных идентификаторов: NewStr, Var, x и т.д.

Примеры неправильных идентификаторов: Check!, 1var, ab\_c и т.д.

## **Литералы**

Литерал — запись в исходном коде компьютерной программы, представляющая собой фиксированное значение. Литералами также называют представление значения некоторого типа данных.

В языке MAV-2020 существует 2 вида литералов: литералы целого типа и строковые, описаны в таблице 1.3.

Таблица 1.3 – Литералы

|  |  |
| --- | --- |
| Тип | Описание |
| Литералы целого типа | Интерпретируются как int, являются rvalue. Задаются в десятичной, восьмеричной, двоичной, шестнадцатеричной форме. |
| Строковые литералы | Интерпретируются как string, заключаются в одинарные кавычки (‘Hello’), являются rvalue |
| Булевые литералы | Интерпретируются как boolean, являются rvalue. Могут быть представлены как true | false |

## **Объявление данных и область видимости**

В языке MAV-2020 в объявлении данных указывается тип данных и имя идентификатора. Требуется обязательное объявление переменной перед её использованием.

Примеры: int a, string b;

Все переменные в языке MAV-2020 имеют область видимости, а именно префикс — название функции, в которой они находятся, что разрешает использование в различных функциях переменных с одинаковым именем. Параметры функции видны только внутри неё. Переменные, объявленные в одной функции, недоступны в другой.

## **Инициализация данных**

В момент объявления переменных в языке MAV-2020 происходит автоматическая инициализация в зависимости от типа данных. Инициализация другими значениями в момент объявления не допускается. Присвоение значения переменным должны происходить после непосредственного объявления. Виды инициализации представлены в таблице 1.4.

Таблица 1.4 — Способы инициализации переменных

|  |  |
| --- | --- |
| Вид инициализации | Примечание |
| <тип данных> <идентификатор>; | Автоматическая инициализация: переменные типа int инициализируются нулём, переменные типа string – пустой строкой, boolean - false. |
| <идентификатор> = <значение>; | Присваивание переменной значения. |

## **Инструкции языка**

Инструкции языка MAV-2020 представлена в таблице 1.5.

Таблица 1.5 – Инструкции языка

|  |  |
| --- | --- |
| Инструкция | Форма записи |
| Объявление переменной | <тип данных> <идентификатор>. |
| Объявление и определение функции | def <тип данных> <идентификатор> (<тип данных> <идентификатор>, …)  {  / программный блок /  return <идентификатор/литерал>.  }; |
| Объявление функции  стандартной библиотеки | spot <тип данных> <идентификатор> (<тип данных> <идентификатор>, …); |
| Вызов функции | <идентификатор> (<идентификатор>, ...) |
| Присвоение значения | = |
| Условный оператор | if <литерал> | <идентификатор>|<логическое выражение>  / программный блок /  elif<литерал>|<идентификатор>|<логическое выражение>  else  / программный блок /  endif |
| Операторы цикла | while<литерал>|<идентификатор>|<логическое выражение>  / программный блок /  endw  repeat  / программный блок /  until<литерал>|<идентификатор>|<логическое выражение> |

Продолжение таблицы 1.5

|  |  |
| --- | --- |
| Вывод данных | s\_out <литерал/идентификатор>. |
| Функции стандартной библиотеки  (применяются при инициализации и в выражениях) | int strtoint(string) — преобразовывает строку в число  int strlength (string) — находит длину строки  int std\_in\_int(int) – ввод целочисленного значение и его возвращение. |
| Возвращаемое значение | return <литерал | идентификатор>. |

## **Операции языка**

Арифметические операции, которые можно использовать в языке MAV-2020, представлены в таблице 1.6.

Таблица 1.6 — Операции языка

|  |  |
| --- | --- |
| Инструкция | Форма записи |
| Арифметические операции языка | + — бинарное сложение  − — бинарный минус  \* — бинарное произведение  / — бинарное деление  % — бинарное взятие остатка  ( ) — приоритет операций |
| Операции сравнения | more\_than — больше  less\_than — меньше  equals — равно  not\_equals — не равно  less\_equals — меньше или равно  more\_equals — больше или равно |

Приоритет подчиняется правилам:

* если в вы­ра­же­ние без ско­бок вхо­дят толь­ко сло­же­ние и вы­чи­та­ние или толь­ко умно­же­ние и де­ле­ние, то дей­ствия вы­пол­ня­ют в том по­ряд­ке, в каком они на­пи­са­ны;
* если в вы­ра­же­ние без ско­бок вхо­дят не толь­ко дей­ствия сло­же­ния и вы­чи­та­ния, но и умно­же­ния и де­ле­ния, или оба этих дей­ствия, то сна­ча­ла вы­пол­ня­ют по по­ряд­ку (слева на­пра­во) умно­же­ние и де­ле­ние, а затем сло­же­ние и вы­чи­та­ние;
* если в вы­ра­же­нии име­ют­ся скоб­ки, то сна­ча­ла вы­чис­ля­ют зна­че­ние вы­ра­же­ний в скоб­ках.

## **Выражения и их вычисления**

Правила для выражения и их вычисления в языке MAV-2020:

* выражение записывается только в одну строку;
* допускается использование скобок (для изменения приоритета);
* не допускается запись двух подряд идущих арифметических операций;
* выражение может содержать вызов функции.

## **Программные конструкции языка**

Основные программные конструкции языка MAV-2020 представлены в таблице 1.7.

Таблица 1.7 — Основные конструкции языка

|  |  |
| --- | --- |
| Конструкция | Реализация |
| Главная функция  (точка входа) | main  {  / программный блок /  } |
| Функции | def <тип данных> <идентификатор>  (<тип данных> <идентификатор>, …)  {  / программный блок /  return <идентификатор/литерал>;  }  def <тип данных> <идентификатор> ()  {  return <идентификатор/литерал>.  }  spot <тип данных> <идентификатор>  (<тип данных> <идентификатор>, …); |

## **Область видимости идентификаторов**

Область видимости в языке MAV-2020 - «сверху вниз» (по принципу С++). Это значит, что к каждому идентификатору можно обратиться только ниже описания его описания или объявления. «Ниже» понимается буквально — ниже по тексту программы. Т.е., например, до описания функции ее нельзя вызывать.

Все переменные в языке MAV-2020 обязаны находится внутри программного блока функций. Объявление глобальных переменных (вне программных блоков функций) не предусмотрено.

## **Семантические проверки**

Основные семантические правила языка MAV-2020 проверяемые на этапах работы транслятора, представлены в таблице 1.8.

Таблица 1.8 — Семантические правила

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Должна присутствовать точка входа main и только одна |
| 2 | Идентификаторы должны быть объявлены до инициализации и использования |
| 3 | Не должно быть объявлений идентификаторов с одинаковыми именами в одном и том же блоке кода |
| 4 | Присваивать значение идентификатору можно только соответствующего типа |
| 5 | Вызов функции обязует использование скобок после ее названия с передачей параметров соответствующих типов или без них |
| 6 | Тип возвращаемого функцией значения должен соответствовать типу функции |
| 7 | Деление на ноль запрещено |
| 8 | Присваивать значение функции запрещено |
| 9 | Проводить арифметические операции со строковым типом данных запрещено |

## **Распределение оперативной памяти на этапе выполнения**

На этапе выполнения все переменные данного языка помещаются в стек.

## **Стандартная библиотека и ее состав**

Функции данной стандартной библиотеки применяются при инициализации и в выражениях. Описание функций стандартной библиотеки языка MAV-2020 представлено в таблице 1.9.

Таблица 1.9 — Библиотека

|  |  |
| --- | --- |
| Функция | Описание |
| int strtoint (идентификатор | литерал); | Переводит строковый литерал или строковую переменную. Применима для строк, представляющий десятичное целое число. |
| int strlength (идентификатор | литерал); | Получение длины строки. Применима как для строковых литералов, так и для строковых переменных |
| int std\_in\_int(идентификатор | литерал); | Вызывает ввод целочисленного значение в консоль, возвращает введённое число. |

## **Ввод и вывод данных**

Ввод данных языком программирования MAV-2020 поддерживается только для целочисленных данных с помощью функции стандартной библиотеки int std\_in\_int(int);.

Для вывода данных используется функция s\_out (<имя идентификатора>).

Пример: s\_out a;

* 1. **Точка входа**

Функция точки входа в языке программирования MAV-2020 представлена в таблице 1.10.

Таблица 1.10 — Точка входа

|  |  |
| --- | --- |
| Конструкция | Реализация |
| Главная функция  (точка входа) | main  {  / программный блок /  } |

## **Препроцессор**

Препроцессор, принимающий и выдающий некоторые данные на вход транслятору, в языке MAV-2020 отсутствует.

## **Соглашения о вызовах**

Соглашение о вызовах — это протокол для передачи аргументов функциям. Другими словами, это договоренность между вызывающим и вызываемым кодом.

В языке MAV-2020 по умолчанию применяется соглашение \_stdcall, где параметры помещаются в стек, передача параметров происходит справа налево, стек освобождает вызываемый код, возврат через регистр EAX.

## **Объектный код**

Исходный код, написанный на языке MAV-2020, транслируется нашим компилятором в язык MASM32.

## **Классификация сообщений транслятора**

В ходе своей работы, транслятор 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 1.12.

Таблица 1.12 – Сообщения транслятора

|  |  |
| --- | --- |
| Интервал | Описание |
| 0-9 | Ошибки системы |
| 10-100 | Ошибки входных параметров |
| 101-119 | Ошибки открытия и чтения файлов |
| 120-170 | Ошибки на этапе лексического анализа |
| 171-599 | Ошибки на этапе синтаксического анализа |
| 600-999 | Ошибки на этапе семантического анализа |

## **Контрольный пример**

Контрольный пример языка программирования MAV-2020 представлен в приложении А.

# **Глава 2 Структура транслятора**

## **Компоненты транслятора их назначение и принципы взаимодействия**

Транслятор языка MAV-2020 преобразует исходный код программы в код целевого языка. Процесс трансляции состоит из фаз: лексический анализ, синтаксический анализ, семантический анализ и генерация кода. Алгоритм выполнения и описание каждой фазы представлено в таблице 2*.*1. На всех фазах трансляции применяется таблица идентификаторов и таблица лексем. Графически схема транслятора представлена на рисунке 2.1.

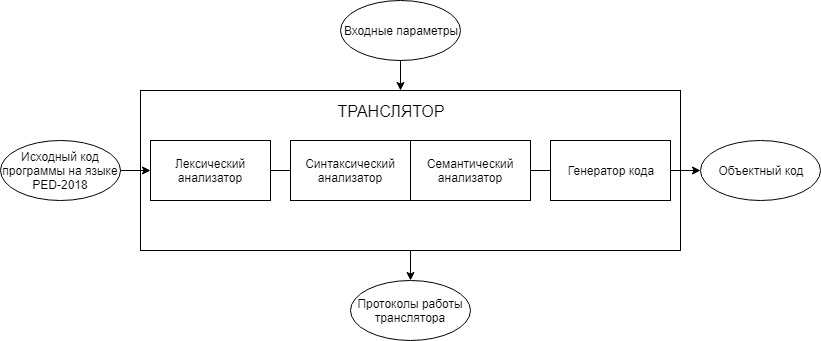


Рисунок 2.1 — Схема структуры транслятора

Таблица 2.1 — Структура транслятора

|  |  |
| --- | --- |
| Наименование подпрограммы | Предназначение |
| Лексический анализатор | Обрабатывает входной файл исходного кода, проверяя его на разрешённые, запрещённые и игнорируемые символы. Преобразует исходный код в более простой с помощью замены длинных слов (идентификаторов, типов данных, ключевых слов и т.д.) на лексемы, состоящие из одного символа, что упрощает последующую работу с кодом. После обработки на формируется таблица лексем и таблица идентификаторов. |

Продолжение таблицы 2.1

|  |  |
| --- | --- |
| Синтаксический анализатор | Проверяет правильность написанных конструкций по заданной грамматике, выявляет синтаксические ошибки при их наличии и формирует дерево разбора. |
| Семантический анализатор | Проверяет правильность исходного кода с точки зрения семантики. Как отдельная программа в данном трансляторе не реализована, вся проверка по правилам осуществляется на этапах лексического и синтаксического анализа. |
| Генератор кода | Выполняющий генерацию кода на языке MAV-2020 в код на языке MASM на основе данных, полученных на предыдущих этапах трансляции, а именно таблицы лексем и таблицы идентификаторов. |

## **Перечень входных параметров транслятора**

Благодаря входным параметрам транслятора осуществляется контроль за выводом протоколов работы разных компонент, а также указывается файл с исходным кодом — программы. Перечень входных параметров представлена в таблице 2.2.

Таблица 2.2 — Входные параметры транслятора

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Тип параметра |
| -in: | Указывает транслятору путь к исходному коду. | Обязательный. |
| -log: | Указывает транслятору в какой файл выводить протокол работы транслятора. | Не обязательный (При отсутствии за основу берется параметр -in). |
| -asm: | Указывает транслятору в какой файл выводить результат работы транслятора. | Не обязательный  (При отсутствии за основу берется параметр -in). |

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

Транслятор языка MAV-2020 формирует один протокол его работы. Информация, записываемая в протокол, представлена в таблице 2.3.

Таблица 2.3 — Протокол транслятора.

|  |  |
| --- | --- |
| Тип информации | Описание  Информации |
| Дата и время трансляции | Выводится дата и время. |
| Параметры командой строки | Выводится информация об указанных параметрах командной строки. |
| Полная таблица лексем | Выводится таблица лексем с информацией к каждой лексеме. |
| Таблица идентификаторов | Выводится таблица идентификаторов с дополнительной информацией. |
| Промежуточный код | Выводится исходный код, представленный в виде лексем. |
| Трассировочная информация синтаксического анализа | Выводится полная информация о разборе таблицы лексем на синтаксическом анализаторе. |
| Дерево разбора | Выводится правила, по которым осуществился разбор исходного кода. |
| Промежуточный код с использованием польской записи | Выводится исходный код, представленный в виде лексем, выражения представлены в польской записи. |

# **Глава 3 Разработка лексического анализатора**

## **Структура лексического анализатора**

Лексический анализатор — часть транслятора, выполняющая лексический анализ, т.е. преобразующая исходный текст, заменяя лексические единицы языка их внутренним представлением — лексемами. Входными данными для лексического анализатора является предварительно обработанный текст программы на языке MAV-2020. В результате работы лексического анализатора формируется таблица идентификаторов и таблица лексем, модифицируется протокол работы транслятора. Структура лексического анализатора MAV-2020 представлена на рисунке 3.1.



Рисунок 3.1 — Структура лексического анализатора MAV-2020

* 1. **Контроль входных символов**

Таблица для контроля входных символов представлена на рисунке 3.2.

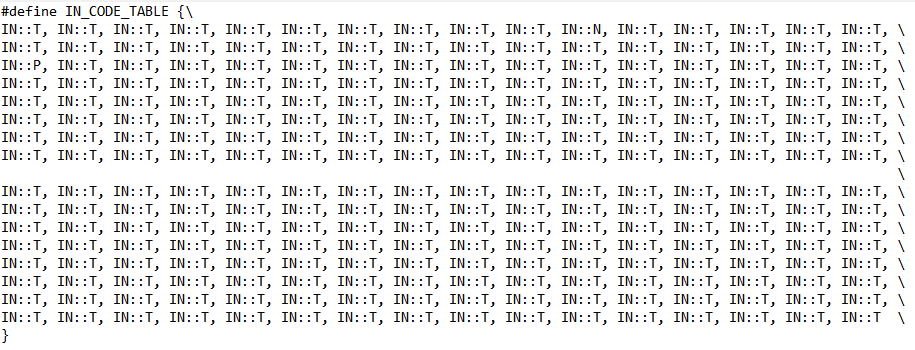


Рисунок 3.2 — Структура лексического анализатора MAV-2020

Принцип работы таблицы: каждому элементу соответствует значение в шестнадцатеричной системе счисления - такое же, как и в таблице Windows-1251 (см. рисунок 1.1).

В представленной таблице: F — запрещённый символ, T — разрешённый символ, I — игнорируемый символ, P — пробел,

N — новая строка

## **Удаление избыточных символов**

Избыточными символами являются символы табуляции и пробелы.

Избыточные символы удаляются на этапе разбиения исходного кода на слова.

Описание алгоритма удаления избыточных символов, а также разбиения исходного кода на цепочки:

* файл с исходным кодом программы считывается посимвольно;
* в конце каждой итерации посимвольного считывания текущий символ сохраняется в буфер;
* встреча пробела включает проверку предыдущего (буферного) символа;
* если предыдущий символ является допустимым символом, текущий пробел — разделитель единиц языка, следовательно, сохраняем считанную единицу в формируемую структуру первично обработанного исходного кода;
* если предыдущий символ является пробелом, текущий символ игнорируется;
* если текущий символ — кавычка, отключается игнорирование пробелов до тех пор, пока не встретится еще одна кавычка (т.к. внутри кавычек содержатся данные пользователя — литералы);

## **Перечень ключевых слов, сепараторов, символов операций соответствующим им лексемам и конечных автоматов**

Ключевые слова языка MAV-2020: int, string, def, return, main, out, strlength, strtoint.

Для каждого слова, операторов и сепараторов определен конечный автомат, по которому происходит разбор выражения. 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.

Благодаря замене цепочек, написанных на языке MAV-2020, лексемами, упрощается дальнейшая обработка исходного кода программы. Перечень цепочек, соответствующих им автоматов и лексем, представлена в приложении Б.

* 1. **Основные структуры данных**

Описание основных структур данных, используемых для хранения таблиц лексем, представлено на рис. 3.3.

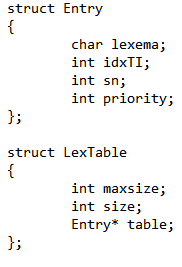


Рисунок 3.3 — Структуры таблиц лексем MAV-2020

Описание основных структур данных, используемых для хранения таблиц идентификаторов, представлено на рис. 3.4.

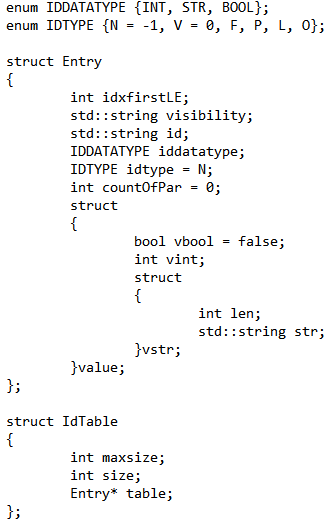


Рисунок 3.4. — Структуры таблиц идентификаторов MAV-2020

## **Принцип обработки ошибок**

При обнаружении ошибки во время работы транслятора, вызывается функция получения ошибки, в которую передается, в зависимости от места возникновения ошибки, следующая информация: код ошибки, номер строки в коде и сообщение о типе ошибки. При возникновении ошибки работа транслятора прекращается. Информация об ошибке выводится в \*.txt.log файл.

## **Структура и перечень сообщений лексического анализатора**

Перечень сообщений, формируемых лексическим анализатором в ходе своей работы, представлен в таблице 3.2.

Таблица 3.2 — Перечень сообщений лексического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 160 | Превышен максимальный размер таблицы идентификаторов |
| 121 | Таблица лексем переполнена |
| 108 | Ошибка лексического анализа |

## **Параметры лексического анализатора и режимы его работы**

Транслятор допускает использование параметра для управления работой лексического анализатора, а именно выводом таблицы лексем и таблицы идентификаторов. Описание параметров представлено в таблице 2.2.

## **Алгоритм лексического анализа**

Лексический анализ является первой и наиболее простой фазой трансляции. Алгоритм лексического анализатора заключается в следующем: после разбиения текста из файла с исходным кодом на слова во множество в виде пары ключ – значение, где ключ – порядковый номер слова, значение – строковое представление слова и номер его строки в исходном коде. Для этого используется структура данных из STL (Стандартная библиотека шаблонов C++).

template<class Key,  
    class T,  
    class Compare = [std::less](http://en.cppreference.com/w/cpp/utility/functional/less)<Key>,  
    class Allocator = [std::allocator](http://en.cppreference.com/w/cpp/memory/allocator)<[std::pair](http://en.cppreference.com/w/cpp/utility/pair)<const Key, T> >

>class map;

Анализ ключевых слов языка осуществляется с помощью вызова перегруженного оператора сравнения для значения, находящегося во множестве. Для анализа имён идентификаторов, строковых и целочисленных операторов применяются регулярные выражения std::regex, используя модифицированную грамматику регулярных выражений ECMAScript.



Рисунок 3.5 — Регулярное выражение для целочисленного литерала

## **Контрольный пример**

Результатом работы лексического анализатора являются таблица лексем и таблица идентификаторов. Содержимое таблиц на основе исходного кода из приложения А представлено в приложении Б.

# **Глава 4 Разработка синтаксического анализатора**

## **Структура синтаксического анализатора**

Синтаксический анализатор является второй фазой работы транслятора. Назначением синтаксического анализаторы является распознавание синтаксических конструкций языка и формирование промежуточного кода. Исходными данными синтаксического анализатора являются таблицы лексем и идентификаторов. Свою работу начинает только при условии отсутствии ошибок в их общей структуре. Лексемы являются для синтаксического анализатора терминальными символами контекстно-свободной грамматики. Если в ходе его работы не было обнаружено ошибок, то формируется дерево разбора (промежуточное представление кода) и модифицируется протокол работы. Структура синтаксического анализатора представлена на рисунке 4.1.



Рисунок 4.1 — Структура синтаксического анализатора MAV-2020

## **Контекстно-свободная грамматика, описывающая синтаксис языка**

Каждый язык программирования описывается с помощью набора правил, определяющих структуру правильной программы. Наиболее удобным формализмом для описания синтаксических конструкций языка программирования являются контекстно-свободные грамматики.

Грамматика для синтаксического разбора языка MAV-2020 представляется четверкой G = <T, N, P, S>, где Т — множество терминальных символов, N — множество нетерминальных символов, P — множество правил языка, S — начальный символ грамматики, являющийся нетерминалом.

В грамматике языка MAV-2020 множество нетерминальных символов представлено следующим образом:

* S — Порождает правила, описывающие общую структуру программы;
* N — Порождает правила, описывающие основные конструкции языка;
* Е — Порождает правила, описывающие выражения;
* F — Порождает правила, описывающие формальные параметры функции;
* M — Порождает правила, описывающие арифметические действия;
* W — Порождает правила, описывающие фактические параметры функции;
* P – Порождает правила описывающие логические выражения
* T – Порождает правила описывающие условный оператор elif

Перечень правил, описывавших контекстно-свободную грамматику языка MAV-2020, представлен в таблице 4.1.

Таблица 4.1 – Правила грамматики MAV-2020

|  |  |
| --- | --- |
| Нетерминальный символ | Цепочки правил |
| S→ | m{N} | fti(F){NrE;};S} | fti(F);S| fti(F){N}rE;} |
| N→ | ti; | i=E; | rE; | ?PNn | ?PNeNnN | ti;N | rE;N | i=E;N | pE; | pE;N  !NuP | ~PN\* | !NuPN | ~PN\*N | ?PNTn | ?PNTnN | ?PNTnN | ?PNTn | ?PNTeNn | ?PNTeNnN |
| E→ | i | l | (E) | i(W) | i() | iM | lM | (E)M | i(W)M |
| F→ | ti | ti, F |
| W→ | i | l | i,W | l,W |
| M→ | vE | vEM |
| P → | I | l | iqi | iql | lqi |
| T → | ^PN | ^PNT |

## **Построение конечного магазинного автомата**

Конечный автомат с магазинной памятью представляет собой семерку,где Q — множество состояний автомата, V — алфавит входных символов, Z – алфавит специальных магазинных символов,  — функция переходов автомата,  — начальное состояние автомата,  — начальное состояние магазинного автомата, F — множество конечных состояний.

Принцип работы автомата следующий:

* в магазин записывается стартовый символ;
* на основе полученных раннее 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 пункту 4;
* если наш символ достиг дна стека, и лента в этот момент пуста, то синтаксический анализ выполнен успешно. Иначе вызывается ошибка.

Пример работы магазинного автомата для цепочки представлен на рисунке 4.2.

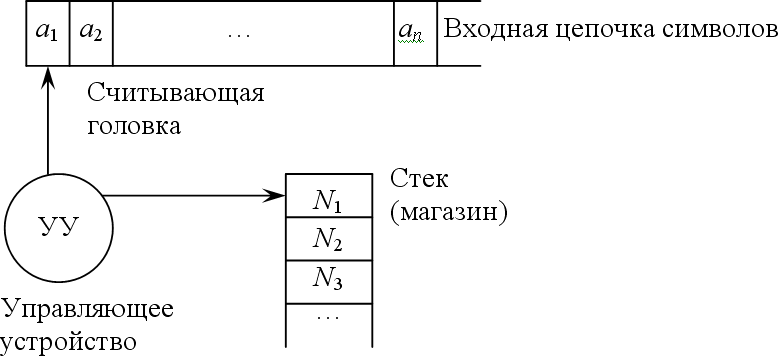


Рисунок 4.2 — Автомат с магазинной памятью

## **Основные структуры данных**

Основные структуры данных синтаксического анализатора включают в себя структуру магазинного автомата и структуру грамматики Грейбах, описывающая правила языка MAV-2020. Структура синтаксического анализатора представлена в приложении Г.

## **Описание алгоритма синтаксического разбора**

Алгоритм синтаксического разбора:

* происходит поиск и выделение синтаксических конструкций в исходном тексте (разбор);
* распознавание (проверка правильности) синтаксических конструкций;
* выявление ошибок и продолжение процесса распознавания после обработки ошибок;
* в случае отсутствия ошибок, формируется дерево разбора.

## **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 |
| 170 | Неверная структура программы |
| 171 | Ошибочный оператор |
| 172 | Ошибка в выражении |
| 173 | Ошибка в параметрах функции |
| 174 | Ошибка в параметрах вызываемой функции |
| 175 | Ошибочное выражение |
| 176 | Ошибка в логическом выражении |
| 177 | Ошибка в условном операторе |

## **Параметры синтаксического анализатора и режимы его работы**

Транслятор допускает использование параметра для управления работой синтаксического анализатора, а именно вывода дерева разбора и трассировки. Описание параметров представлено в таблице 2.2.

## **Принцип обработки ошибок**

Принцип заключается в том, что синтаксический анализатор перебирает все возможные правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем. В случае если не была найдена ни одна подходящая цепочка, то формируется соответствующая ошибка из таблицы 4.2. Все ошибки записываются в общую структуру ошибок, а также отображаются на консоли.

## **Контрольный пример**

Из контрольного примера, представленного в приложении А, результатом работы синтаксического анализатора, является трассировка и дерево разбора, представленные в приложении В.

# **Глава 5 Разработка семантического анализатора**

## **Структура семантического анализатора**

Семантический анализатор в трансляторе языка MAV-2020 не выделен в отдельную фазу. Проверка на ошибки в исходном коде производится на этапе лексического анализа.

## **Функции семантического анализа**

Основные действия семантического анализатора:

* проверка семантических правил исходного языка;
* дополнение внутреннего представления программы операторами и действиями, неявно предусмотренными семантикой исходного языка.

Семантические правилаязыка MAV-2020 представлены в гл. 1. п. 1.16.

## **5.3 Структура и перечень сообщений семантического анализатора**

Сообщения, формируемые семантическим анализатором представлены в таблице 5.1.

Таблица 5.1 — Перечень сообщений семантического анализатора

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 600 | Попытка переопределения идентификатора |
| 601 | Отсутствует точка входа main |
| 602 | Точка входа main должна быть единственной |
| 603 | Несоответствие типов операндов |
| 604 | Арифметические операции со строками недопустимы |
| 605 | Деление на 0 |
| 606 | Тип функции и возвращаемой переменной не совпадают |
| 607 | Количество параметров в вызываемой функции не соответствует её сигнатуре |
| 608 | Ошибочная запись строкового литерала |
| 609 | Функции стандартной библиотеки не могут иметь определения |
| 610 | Объявление функции стандартной библиотеки должно осуществляться с помощью ключевого слово spot |

## **5.4 Принцип обработки ошибок**

Принцип обработки ошибок идентичен принципу обработки ошибок на этапе лексического анализа (раздел 3.6).

## **5.5 Контрольный пример**

Контрольный пример заключается в тестировании функций семантического анализатора при наличии соответствующих ошибок в исходном коде. Тестирование представлено в таблице 5.3.

Таблица 5.3 – Тестирование функций

|  |  |
| --- | --- |
| Исходный код с ошибкой | Генерируемое сообщение об ошибке |
| main  {  k = 5;  int k;  s\_out k;  } | Ошибка: 611: Переменная должна иметь определенный тип данных  Строка: 2 |
| main  {  string f;  f = 'hi';  s\_out f;  }  main  {  int l;  l = 12;  } | Ошибка: 602: Точка входа main должна быть единственной  Строка: 6 |
| main  {  int k;  k = 5/0;  s\_out k  } | Ошибка: 605: Деление на 0  Строка: 3 |

# **Глава 6 Вычисление выражений**

## **Выражения, допускаемые языком**

Выражения, допускаемые языком MAV-2020, выполняются над целочисленными типами данных. В выражениях поддерживаются арифметические операции, такие как +, -, \*, / и (), и вызовы функций из них.

Приоритетность арифметических операций представлена в таблице 6.1.

Таблица 6.1 — Приоритетность операций

|  |  |  |
| --- | --- | --- |
| Операция | Приоритет | Порядок выполнения |
| ( | 1 | Наивысший приоритет (выполняется в первую очередь) |
| ) | 1 | Наивысший приоритет (выполняется в первую очередь) |
| \* | 2 | Вторая степень приоритетности |
| / | 2 | Вторая степень приоритетности |
| % | 2 | Вторая степень приоритетности |
| + | 3 | Наименьший приоритет (выполняется в последнюю очередь) |
| - | 3 | Наименьший приоритет (выполняется в последнюю очередь) |

## **Польская запись и принцип ее 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.

Обратная польская запись — это форма записи математических выражений, в которой операторы расположены после своих операндов. Выражение в обратной польской нотации читается слева направо: операция выполняется над двумя операндами, непосредственно стоящими перед знаком этой операции. Результат операции заменяет в выражении последовательность её операндов и символ операции. Результатом вычисления всего выражения является результат последней вычисленной операции.

Алгоритм построения польской записи:

* исходная строка: выражение;
* результирующая строка: польская запись;
* стек: пустой;
* исходная строка просматривается слева направо;
* операнды переносятся в результирующую строку;
* операция записывается в стек, если стек пуст;
* операция выталкивает все операции с большим или равным приоритетом в результирующую строку;
* отрывающая скобка помещается в стек;
* закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются.
* по концу разбора исходной строки все операции, оставшиеся в стеке, выталкиваются в результирующую строку.

## **Программная реализация обработки выражений**

Программная реализация алгоритма преобразования выражений в обратный польский формат основана на функциях, представленных на рисунке 6.1. Изначально происходит поиск выражения в таблице лексем функцией start. Преобразование выражений в польскую запись реализовано с помощью функции PolishNotation, результатом работы которой является промежуточный код, который записывается в общий протокол работы транслятора.

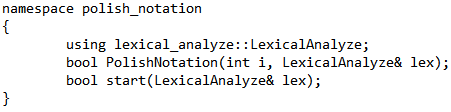


Рисунок 6.1 — Фрагмент кода для преобразования выражений

## **Контрольный пример**

Результаты преобразования выражений исходного кода в польскую запись представлены в приложении A. Ход разбора одного из выражений в контрольном примере в польский формат записи приведен в таблице 6.2.

Таблица 6.2 — Пример конвертации выражения в польскую запись

|  |  |  |
| --- | --- | --- |
| Выражение | Стек | Результат |
| 3\*(5-2)+3/6 |  |  |
| \*(5-2)+3/6 |  | 3 |
| (5-2)+3/6 | \* | 3 |
| 5-2)+3/6 | \*( | 3 |
| -2)+3/6 | \*( | 35 |
| 2)+3/6 | \*(- | 35 |
| )+3/6 | \*(- | 352 |
| +3/6 | \* | 352- |
| 3/6 | **+** | 352-\* |
| /6 | **+** | 352-\*3 |
| 6 | **+/** | 352-\*3 |
|  | **+/** | 352-\*36 |
|  |  | 352-\*36/+ |

# **Глава 7 Генерация кода**

## **7.1 Структура генератора кода**

Генерация объектного кода — это перевод транслятором представления исходной программы на языке MAV-2020 в цепочку символов выходного языка MASM32 под Windows 10. На вход генератора подаются таблицы лексем и идентификаторов, а также исходный код, разбитый на слова, на основе которых генерируется файл с кодом на языке MASM32, который впоследствии будет скомпилирован в загрузочный модуль. Схематично генерация кода показана на рисунке 7.1.Генератор кода начинает свою работу только в том случае, если код на языке MAV-2020 прошёл предыдущие этапы без ошибок.

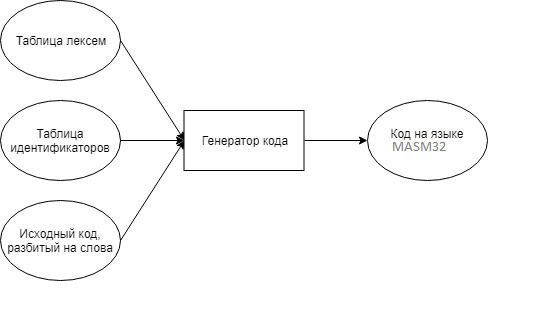


Рисунок 7.1 — Структура генератора кода

## **7.2 Представление типов данных в оперативной памяти**

Язык MAV-2020 требует указывать тип данных при объявлении идентификатора. Язык MASM32 обязывает указывать тип, объявление происходит с указанием типа данных и имени переменной. Соответствия между типами данных идентификаторов языка MAV-2020 и языка MASM32 представлены в таблице 7.1.

Таблица 7.1 — Соответствие типов данных

|  |  |
| --- | --- |
| Тип данных на языке MAV-2020 | Тип данных на языке MASM32 |
| int (целочисленный) | SDWORD |
| string (строковый) | DB (литералы), DWORD(переменные) |
| boolean | DWORD |

## **7.3 Статическая библиотека**

Для языка MAV-2020 статическая библиотека реализована как отдельный файл MAVSTD.lib. Реализация функций стандартной библиотеки была осуществлена с помощью языка C++.

## **7.4 Особенности алгоритма генерации кода**

Алгоритм генерации исходного кода на языке MAV-2020:

* открытие файла “\*.asm” для записи;
* на вход генератора подаются таблицы лексем и идентификаторов, а также исходный код, разбитый на слова;
* каждый элемент таблицы лексем проверяется на соответствие с зарезервированными лексемами;
* если соответствие найдено, в выходной файл “\*.asm” записывается соответствующее выражение. Пример данной операции для 1 лексемы представлен на рисунке 7.2;
* когда каждая лексема из таблицы разобрана, код сгенерирован, файл закрывается;

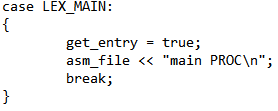


Рисунок 7.2 — Часть кода генерации

## **7.5 Входные параметры генератора кода**

На этапе генерации кода транслятор допускает использование параметра

-asm:. По умолчанию файлом для сгенерированного кода является файл “\*.asm”.

## **7.6 Контрольный пример**

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

# **Глава 8 Тестирование транслятора**

## **8.1 Общие положения**

В результате обработки исходного кода программы, представленного в приложении А, транслятор языка MAV-2020 генерирует общий протокол работы, куда записываются все возникшие ошибки и предупреждения. Кроме того, все ошибки, возникшие на этапах лексического и семантического анализов, выводятся на консоль и в файл. Из ошибок, возникших на этапе синтаксического анализа, на консоль выводится только первая.

## **8.2 Результаты тестирования**

Транслятор языка MAV-2020 представляет диагностику и выявление ошибок на разных этапах трансляции. Ниже будут приведены результаты обработки транслятором исходного кода с допущенными ошибками.

Тестирование ошибок транслятора представлено в таблице 8.1. В таблице 8.1 приведены три вида ошибок, первая происходит на этапе лексического анализа, вторая — синтаксического, третья — семантического.

Таблица 8.1 — Тестирование

|  |  |
| --- | --- |
| Исходный код с ошибкой | Генерируемое сообщение об ошибке |
| main  {  int 8false\_lex;  } | Ошибка: 108: Ошибка лексического анализа,  Строка: 3 позиция 10 |
| main  {  if true  int x;  elif x + 1  string y;  endif  } | Ошибка: 172: Ошибка в выражении  Строка: 4 |
| def int strlength(string x);  main  {  int x;  } | Ошибка: 610: Объявление функции стандартной библиотеки должно осуществляться с помощью ключевого слово spot |

# 

# **Заключение**

В данном курсовом проекте были выполнены не только поставленные минимальные требования. Основной целью курсовой работы было понять принцип работы языков программирования и усвоить эти знания на собственном примере. Данный курсовой проект позволил совместить закрепление знаний сразу по двум языкам программирования, таких как C++ и MASM32 для Windows 10. Кроме того, были усвоены такие понятия как синтаксический, лексический и семантический анализаторы и многие другие.

В итоге был получен примитивный язык программирования MAV-2020, который не имеет сложных конструкций, которые реализованы на сегодняшний день во многих других языках программирования.

Окончательная версия языка MAV-2020 включает:

* 3 типа данных;
* Поддержка операции вывода и ввода;
* Возможность вызова функций стандартной библиотеки;
* Наличие 5 арифметических операторов для вычисления выражений;
* Наличие 6 человеко-ориентированных булевых операторов представленных в виде фраз;
* Структурированная система для обработки ошибок пользователя.

Основные характеристики транслятора MAV-2020:

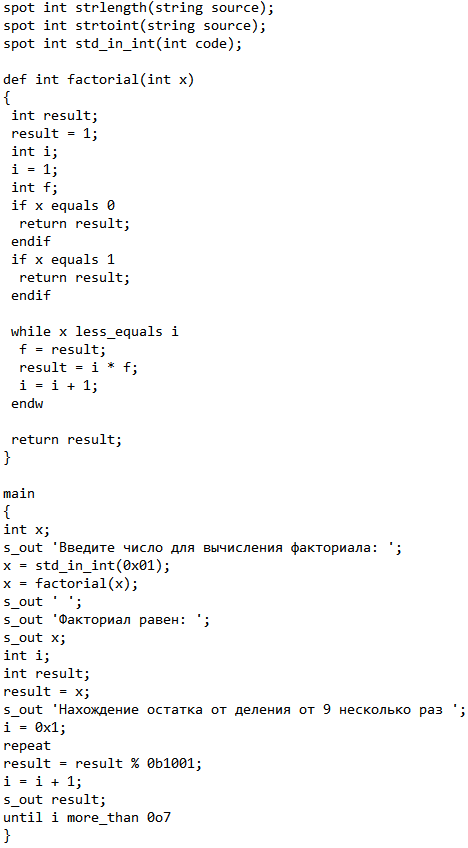
* Возможность обработки 3 входных параметров;
* Реализация 51 цепочки правил грамматики;
* Наличие порядка 3500 строк кода;

# **Список использованных источников**

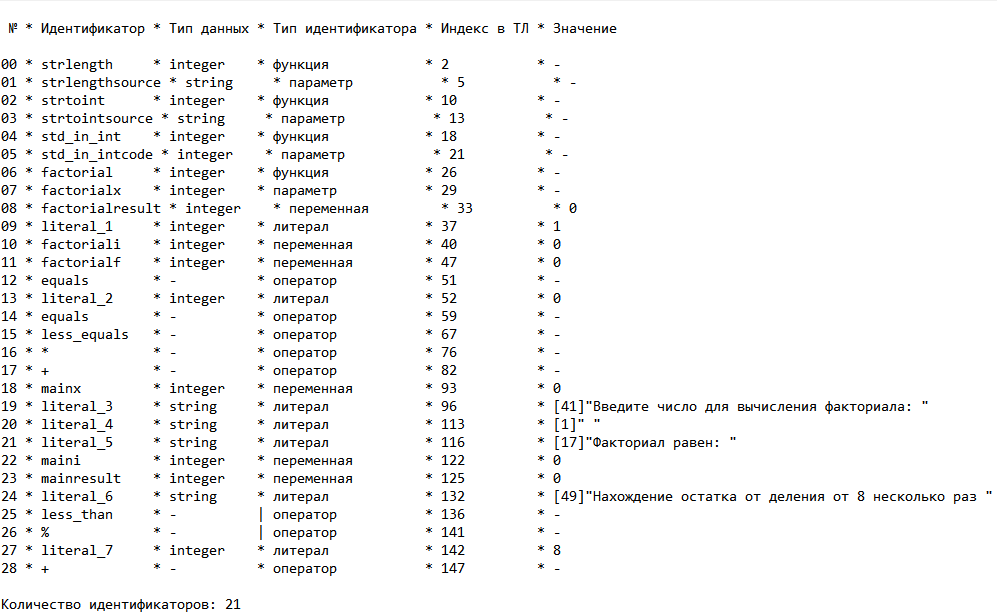
1 [«Всё о Паскале»](http://pascal.net.ru/)  > [Таблица символов 1251 (ANSI, WIN)](http://pascal.net.ru/1251) [Электронный ресурс] – Режим доступа: http://pascal.net.ru/1251

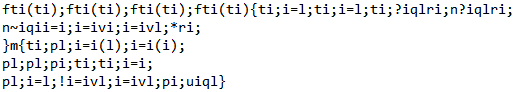
1. Справочник по ассемблеру макросов (Майкрософт) [Электронный ресурс] – Режим доступа: https://docs.microsoft.com/ru-ru/cpp/assembler/masm/microsoft-macro-assembler-reference?view=msvc-160
2. Программирование игр на Ассемблере (часть II) [Электронный ресурс] – Режим доступа: https://www.helloworld.ru/texts/comp/lang/asm/win32/win322.htm.
3. Классификация языков программирования [Электронный ресурс] – Режим доступа: http://bourabai.kz/alg/classification.htm.

# **Приложение А**

****

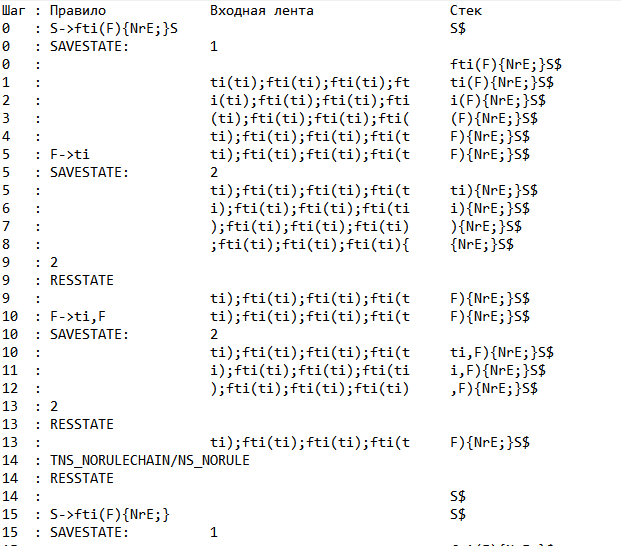
# **Приложение Б**

****

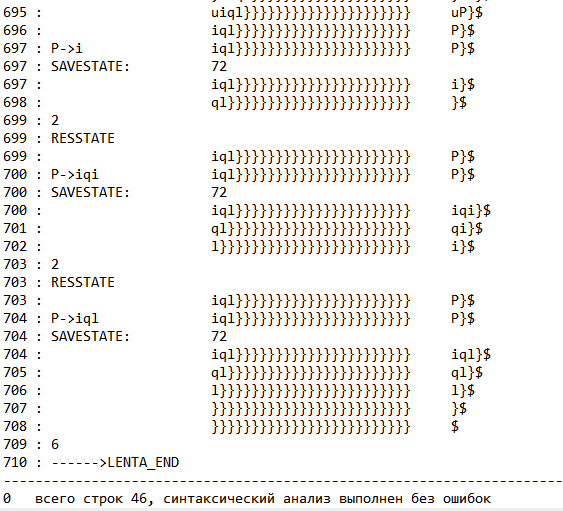
****

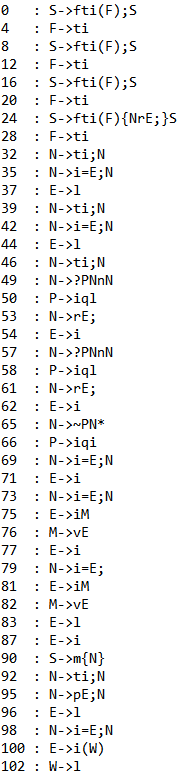
# **Приложение В**

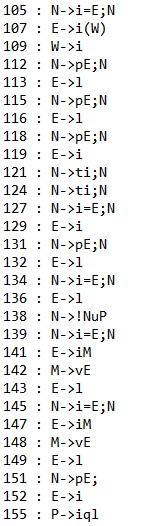
Начало разбора

****

Конец разбора

****

****

****

# **Приложение Г**

