МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ

ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра программной инженерии

Специальность 6-05-0612-01 Программная инженерия

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора BVS-2024»

Выполнил студент Бабич Виолетта Станиславовна

(Ф.И.О.)

Руководитель проекта Волчек Дарья Ивановна

Заведующий кафедрой к.т.н., доц. Смелов В.В.

Консультанты Волчек Дарья Ивановна

Нормоконтролер Волчек Дарья Ивановна

Курсовой проект защищен с оценкой

Минск 2024

**Содержание**

[Введение 3](#_Toc185178121)

[1. Спецификация языка программирования 5](#_Toc185178122)

[1.1 Характеристика языка программирования 5](#_Toc185178123)

[1.2 Определение алфавита языка программирования 5](#_Toc185178124)

[1.3 Применяемые сепараторы 5](#_Toc185178125)

[1.4 Применяемые кодировки 5](#_Toc185178126)

[1.5 Типы данных 5](#_Toc185178127)

[1.6 Преобразование типов данных 6](#_Toc185178128)

[1.7 Идентификаторы 6](#_Toc185178129)

[1.8 Литералы 6](#_Toc185178130)

[1.9 Объявление данных 7](#_Toc185178131)

[1.10 Инициализация данных 7](#_Toc185178132)

[1.11 Инструкции языка 7](#_Toc185178133)

[1.12 Операции языка 8](#_Toc185178134)

[1.13 Выражения и их вычисление 8](#_Toc185178135)

[1.14 Конструкции языка 8](#_Toc185178136)

[1.15 Область видимости идентификаторов 9](#_Toc185178137)

[1.16 Семантические проверки 9](#_Toc185178138)

[1.17 Распределение оперативной памяти на этапе выполнения 10](#_Toc185178139)

[1.18 Стандартная библиотека и ее состав 10](#_Toc185178140)

[1.19 Ввод и вывод данных 10](#_Toc185178141)

[1.20 Точка входа 10](#_Toc185178142)

[1.21 Препроцессор 10](#_Toc185178143)

[1.22 Соглашения о вызовах 10](#_Toc185178144)

[1.23 Объектный код 11](#_Toc185178145)

[1.24 Классификация сообщений транслятора 11](#_Toc185178146)

[1.25 Контрольный пример 11](#_Toc185178147)

[2. Структура транслятора 12](#_Toc185178148)

[2.1 Компоненты транслятора их назначение и принципы взаимодействия 12](#_Toc185178149)

[2.2 Перечень входных параметров транслятора 13](#_Toc185178150)

[2.3 Перечень входных параметров транслятора 13](#_Toc185178151)

[3. Разработка лексического анализатора 14](#_Toc185178152)

[3.1 Структура лексического анализатора 14](#_Toc185178153)

[3.2. Контроль входных символов 14](#_Toc185178154)

[3.3 Удаление избыточных символов 14](#_Toc185178155)

[3.4 Перечень ключевых слов 15](#_Toc185178156)

[3.5 Основные структуры данных 16](#_Toc185178157)

[3.6 Структура и перечень сообщений лексического анализатора 17](#_Toc185178158)

[3.7 Принцип обработки ошибок 18](#_Toc185178159)

[3.8 Параметры лексического анализатора 18](#_Toc185178160)

[3.9 Алгоритм лексического анализа 18](#_Toc185178161)

[3.10 Контрольный пример 18](#_Toc185178162)

[4. Разработка синтаксического анализатора 19](#_Toc185178163)

[4.1 Структура синтаксического анализатора 19](#_Toc185178164)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 19](#_Toc185178165)

[4.3 Построение конечного магазинного автомата 20](#_Toc185178166)

[4.4 Основные структуры данных 21](#_Toc185178167)

[4.5 Описание алгоритма синтаксического разбора 22](#_Toc185178168)

[4.6 Структура и перечень сообщений синтаксического анализатора 22](#_Toc185178169)

[4.7 Параметры синтаксического анализатора и режимы его работы 23](#_Toc185178170)

[4.8 Принцип обработки ошибок 23](#_Toc185178171)

[4.9 Контрольный пример 23](#_Toc185178172)

[5. Разработка семантического анализатора 24](#_Toc185178173)

[5.1 Структура семантического анализатора 24](#_Toc185178174)

[5.2 Функции семантического анализатора 24](#_Toc185178175)

[5.3 Структура и перечень сообщений семантического анализатора 24](#_Toc185178176)

[5.4 Принцип обработки ошибок 25](#_Toc185178177)

[5.5 Контрольный пример 25](#_Toc185178178)

[6. Вычисление выражений 26](#_Toc185178179)

[6.1 Выражения, допускаемые языком 26](#_Toc185178180)

[6.2 Польская запись и принцип ее построения 26](#_Toc185178181)

[6.3 Программная реализация обработки выражений 27](#_Toc185178182)

[6.4 Контрольный пример 27](#_Toc185178183)

[7. Генерация кода 28](#_Toc185178184)

[7.1 Структура генератора кода 28](#_Toc185178185)

[7.2 Представление типов данных в оперативной памяти 28](#_Toc185178186)

[7.3 Статическая библиотека 29](#_Toc185178187)

[7.4 Особенности алгоритма генерации кода 29](#_Toc185178188)

[7.5 Входные параметры, управляющие генерацией кода 29](#_Toc185178189)

[7.6 Контрольный пример 31](#_Toc185178190)

[8. Тестирование транслятора 32](#_Toc185178191)

[8.1 Общие положения 32](#_Toc185178192)

[8.1 Результаты тестирования 32](#_Toc185178193)

[Заключение 33](#_Toc185178194)

[Список использованных источников 34](#_Toc185178195)

[Приложение А 35](#_Toc185178196)

[Приложение Б 37](#_Toc185178197)

[Приложение В 38](#_Toc185178198)

[Приложение Г 41](#_Toc185178199)

[Приложение Д 42](#_Toc185178200)

[Приложение Е 44](#_Toc185178201)

[Приложение Ж 45](#_Toc185178202)

[Приложение И 46](#_Toc185178203)

# **Введение**

В данном курсовом проекте поставлена задача разработки собственного языка программирования и транслятора для него. Название языка – BVS-2024. Написание транслятора будет осуществляться на языке C++, при этом код на языке BVS-2024 будет транслироваться в язык ассемблера.

Транслятор BVS-2024 состоит из следующих частей:

– лексический и семантический анализаторы;

– синтаксический анализатор;

– генератор исходного кода на языке ассемблера.

Задание на курсовой проект можно разделить на следующие задачи:

– разработка спецификации языка BVS-2024;

– разработка лексического анализатора;

– разработка синтаксического анализатора;

– разработка семантического анализатора;

– разбор арифметических выражений;

– разработка генератора кода;

– тестирование транслятора.

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

# **1. Спецификация языка программирования**

## **1.1 Характеристика языка программирования**

Язык BVS-2024 является компилируемым, строго типизированным, процедурным, поддерживающим парадигму структурного программирования.

## **1.2 Определение алфавита языка программирования**

В языке BVS-2024 разрешено использовать латинские символы [A- Z, a- z], арабские цифры [0-9], а также сепараторы и непечатные символы пробела, перевода строки и табуляции. Символы русского языка разрешены только в строковых литералах.

## **1.3 Применяемые сепараторы**

Символы-сепараторы служат в качестве разделителей конструкций языка во время обработки исходного текста программы с целью разделения на токены.

Применяемые сепараторы в языке BVS-2024, 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 |
| >, <, >=, <=, ==, != | Операции сравнения |

## **1.4 Применяемые кодировки**

Набор символов – таблица, задающая кодировку конечного множества символов алфавита. Такая таблица сопоставляет каждому символу последовательность длиной в один или несколько символов другого алфавита.

В языке BVS-2024 используется кодировка Windows-1251.

## **1.5 Типы данных**

В языке программирования BVS-2024 предусмотрены пять типов данных: беззнаковый целый, символьный, строковый, логический, тип, который ничего не возвращает (аналог void). Описание всех типов представлено в таблице 1.2.

Таблица 1.2 – Типы данных языка BVS-2024

|  |  |  |  |
| --- | --- | --- | --- |
| Название типа данных | Описание типа данных | Размерность | Область значений |
| hallow | Тип, который ничего не возвращает (аналог void) | Не имеет размера | Не имеет размера |
| usint | Беззнаковый целочисленный тип | 4 байта | От 0 до 4294967295 |
| boolean | Логический тип данных. Хранит значения true или false | 1 байт | true (1), false (0) |
| text | Строковый тип данных | 254 байт | Строки длиной до 254 символов |
| symbol | Символьный тип данных. Хранит один символ или его числовое представление | 1 байт | Ограничен одним символом |

## **1.6 Преобразование типов данных**

Преобразование типов данных в языке BVS-2024 не предусматривается, так как язык является строго типизированным.

## **1.7 Идентификаторы**

Идентификатор – это имя, используемое для переменных, функций, параметров.

Идентификаторы могут состоять как из одного, так и из нескольких символов латинского алфавита. В идентификаторе могут быть прописные и строчные буквы латинского алфавита. Идентификаторы не могут совпадать с ключевыми словами.

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

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

## **1.8 Литералы**

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

В языке BVS-2024 существует четыре вида литералов: логические литералы, строковые, символьные, целочисленные. Все литералы и их описание представлено в таблице 1.3.

Таблица 1.3 – Литералы

|  |  |
| --- | --- |
| Тип | Описание |
| Строковые литералы | Интерпретируются как text, заключаются в двойные кавычки (“Hello”). |
| Логические литералы | Применяется в условном операторе. Может быть либо true, либо false. |
| Символьные литералы | Интерпретируются как symbol, заключаются в одиночные кавычки ('A'). |
| Целочисленные литералы | Интерпретируются как usint. Представляют целые числа. |

## **1.9 Объявление данных**

Область видимости «сверху вниз» (по принципу С++). В языке BVS-2024 требуется обязательное объявление переменной перед её использованием, объявление данных начинается с ключевого слова, указания типа данных, затем указывается имя идентификатора.

## **1.10 Инициализация данных**

В момент объявления переменных в языке BVS-2024 происходит автоматическая инициализация в зависимости от типа данных. Виды инициализации представлены в таблице 1.4.

Таблица 1.4 – Способы инициализации переменных

|  |  |
| --- | --- |
| Вид инициализации | Примечание |
| declare <тип данных> <идентификатор>; | Автоматическая инициализация: переменные пустой строкой. |
| <идентификатор>: <литерал>; | Присваивание переменной значения. |

## **1.11 Инструкции языка**

Инструкции языка BVS-2024 представлена в таблице 1.5.

Таблица 1.5 – Инструкция языка

|  |  |
| --- | --- |
| Инструкция | Форма записи |
| Объявление функции | declare <тип данных> function <идентификатор> (<тип данных> <идентификатор>, …)  { / программный блок /  return <идентификатор/литерал>;} |
| Вызов функции | <идентификатор> (<идентификатор>, ...) |
| Присвоение значения | <идентификатор>: <литерал>; |
| Печать данных | display <литерал/идентификатор>; |
| Возвращаемое значение | return <литерал/идентификатор>; |

## **1.12 Операции языка**

Операция – конструкция в языках программирования, аналогичная по записи математическим операциям, то есть специальный способ записи некоторых действий.

В языке BVS-2024 предусмотрены операции, представленные в табл. 1.6. Выполнение операций с разными типами данных не допускается.

Таблица 1.6 – Операции языка

|  |  |  |
| --- | --- | --- |
| Тип операции | Операция | Описание |
| Логические | ==  (usint == usint). | Равенство |
| !=  (usint != usint). | Неравенство |
| <  (usint < usint). | Меньше |
| >  (usint > usint). | Больше |
| >=  (usint > =usint). | Больше или равно |
| <=  (usint < =usint). | Меньше или равно |

## **1.13 Выражения и их вычисление**

Выражение языка программирования BVS-2024 представляет собой совокупность переменных, литералов, вызовов функций, знаков операций, скобок, которая может быть вычислена в соответствии с синтаксисом языка.

* Правила составления выражений:
* Выражения записываются в одну строку;
* В выражении могут присутствовать только операнды одинакового типа;
* В выражении могут использоваться функции. Как стандартные, так и пользовательские;
* В выражениях сравнения допускаются операнды булевого и целочисленного типов.

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

## **1.14 Конструкции языка**

Основные программные конструкции: главная функция (точка входа), условная конструкция, функции языка BVS-2024 представлены в таблице 1.7.

Таблица 1.7 – Конструкции языка

|  |  |
| --- | --- |
| Конструкция | Реализация |
| Главная функция  (точка входа) | main  {  / программный блок /  } |
| Условная конструкция | if(<идентификатор/литерал><условный оператор> <идентификатор/литерал>)  {/ программный блок /} |
| Функции | declare <тип данных> function <идентификатор>  (<тип данных> <идентификатор>, …)  {/ программный блок /  return <идентификатор/литерал>;} |

## **1.15 Область видимости идентификаторов**

Область видимости в языке BVS-2024 - «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 BVS-2024 обязаны находится внутри программного блока функций. Объявление глобальных переменных (вне программных блоков функций) не предусмотрено.

## **1.16 Семантические проверки**

Основные семантические правила языка BVS-2024 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 | При использовании стандартных функций должна быть подключена стандартная библиотека и только один раз |
| 3 | Идентификаторы должны быть объявлены до инициализации и использования |
| 4 | Не должно быть объявлений идентификаторов с одинаковыми именами в одном и том же блоке кода |
| 5 | Присваивать значение идентификатору можно только соответствующего типа |
| 6 | Вызов функции обязует использование скобок после ее названия с передачей параметров соответствующих типов или без них |
| 7 | Тип возвращаемого функцией значения должен соответствовать типу функции |
| 8 | Присваивать значение функции запрещено |

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

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

## **1.18 Стандартная библиотека и ее состав**

Язык BVS-2024 содержит стандартную библиотеку. Подключение стандартной библиотеки обязательно перед использованием функции. Пример функция для данной библиотеки представлена в таблице 1.9.

Таблица 1.9 – Библиотека и ее состав

|  |  |
| --- | --- |
| Функция | Описание |
| StrCmp (string1, string2). | Сравнивает две строки. |
| Sum (number1, number2). | Возвращает сумму двух чисел. |
| Pow (number1, number2) | Возводит число number1 в степень number2 и возвращает результат возведения в степень. |

## **1.19 Ввод и вывод данных**

Ввод данных языком программирования BVS-2024 не поддерживается. Для вывода данных используется функция: display <имя идентификатора>.

## **1.20 Точка входа**

В языке BVS-2024 может быть только одна точка входа и определяется наличием функции main. При инициализации более одной или менее одной – выдаст ошибку лексического анализатора.

## **1.21 Препроцессор**

Препроцессор, принимающий и выдающий некоторые данные на вход транслятору, в языке BVS-2024 отсутствует.

## **1.22 Соглашения о вызовах**

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

В языке BVS-2024 по умолчанию применяется соглашение \_stdcall, где параметры помещаются в стек, передача параметров происходит справа налево, стек освобождает вызываемый код, возврат через регистр EAX.

## **1.23 Объектный код**

Объектный код – подлежащая исполнению форма подходящего представления одного или более процессов (текст программы или язык программы), которая компилируется программирующей системой.

Язык BVS-2024 транслируется в ассемблер, а затем в объектный код.

## **1.24 Классификация сообщений транслятора**

В случае возникновения ошибки в коде программы на языке BVS-2024 и выявления её транслятором в текущий файл протокола выводится сообщение. Их классификация сообщений приведена в таблице 1.10.

Таблица 1.10 – Классификация сообщений транслятора

|  |  |
| --- | --- |
| Диапазон ошибок | Описание |
| 0-99 | Системные ошибки. |
| 110-119 | Ошибки чтения и открытия файлов. |
| 120-199 | Ошибки на лексического анализа. |
| 200-299 | Ошибки синтаксического анализа. |
| 300-399 | Ошибки семантического анализа. |
| 400-999 | Зарезервированные ошибки. |

## **1.25 Контрольный пример**

Контрольный пример языка программирования BVS-2024 представлен в приложении А.

# **2. Структура транслятора**

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

Транслятор – это программа преобразующая исходный код на одном языке программирования в исходный код на другом языке. Схема, поясняющая принцип работы транслятора, изображена на рисунке 2.1.

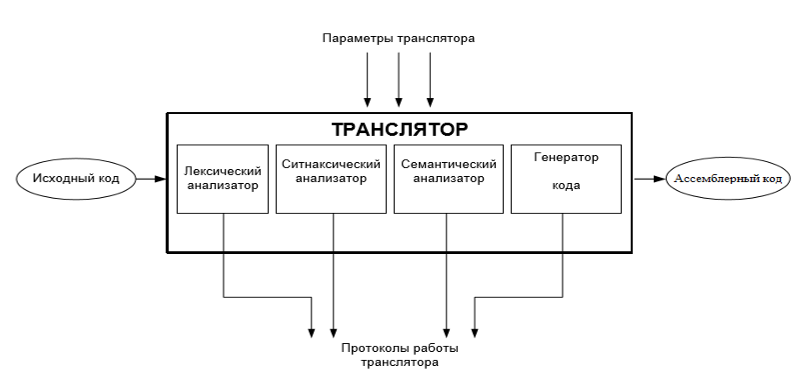


Рисунок 2.1 – Структура транслятора языка BVS-2024

Процесс трансляции исходного кода в язык ассемблера состоит из четырех основных этапов:

* Лексический анализ
* Синтаксический анализ
* Семантический анализ
* Генерация кода

Эти этапы выполняются поочередно, каждый из них обрабатывает данные, полученные от предыдущего, и передает их следующему компоненту транслятора.

Первый этап – лексический анализ. На этом этапе исходный код программы поступает в лексический анализатор, который разделяет его на отдельные токены. Эти токены заменяются на соответствующие лексемы. Результатом работы лексического анализатора являются две таблицы: таблица лексем и таблица идентификаторов.

Следующий этап – синтаксический анализ. Здесь синтаксический анализатор проверяет структуру кода на соответствие синтаксическим правилам языка. На вход подаются таблица лексем и таблица идентификаторов, а результатом работы является дерево синтаксического разбора.

Затем выполняется семантический анализ. Его задача – убедиться, что программа удовлетворяет семантическим правилам языка.

Заключительный этап – генерация кода. Генератор кода на основе таблицы лексем и таблицы идентификаторов создает выходной файл с программой, написанной на языке ассемблера.

## **2.2 Перечень входных параметров транслятора**

Входные параметры необходимы для формирования файлов с результатами работы транслятора. Входные параметры транслятора языка программирования BVS-2024 приведены в таблице 2.1.

Таблица 2.1 – Входные параметры транслятора языка BVS-2024

|  |  |  |
| --- | --- | --- |
| Ключ и входной параметр | Описание | Значение по умолчанию |
| -in:<путь к файлу> | Текстовый файл с исходным кодом на языке BVS-2024. | Отсутствует |
| -out:<путьк файлу> | Выходной файл, содержащий исходный код на языке ассемблера. | Отсутствует |
| -log:<путьк файлу> | Файл с протоколом работы транслятора, таблицей лексем, таблицей идентификаторов, деревом разбора синтаксического анализатора. | <имя файла in>.log |

## **2.3 Перечень входных параметров транслятора**

Протоколы, формируемые транслятором языка BVS-2024 приведены в таблице 2.2.

Таблица 2.2 – Протоколы, формируемые транслятором языка BVS-2024

|  |  |
| --- | --- |
| Протокол | Описание |
| Файл, заданный  параметром “-log:” | Содержит общую информацию о входных параметрах, количестве символов и строк, статус выполнения (успех или ошибка) на каждом этапе, информацию об ошибках, запись ошибок в файл, таблицу лексем, таблицу идентификаторов и дерево разбора. |
| Файл, заданный  параметром “-out:” | Содержит сгенерированный код на языке Ассемблера. |

# **3. Разработка лексического анализатора**

## **3.1 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 BVS-2024. В результате работы лексического анализатора формируется таблица идентификаторов и таблица лексем, модифицируется протокол работы транслятора. Структура лексического анализатора BVS-2024 представлена на рисунке 3.1.

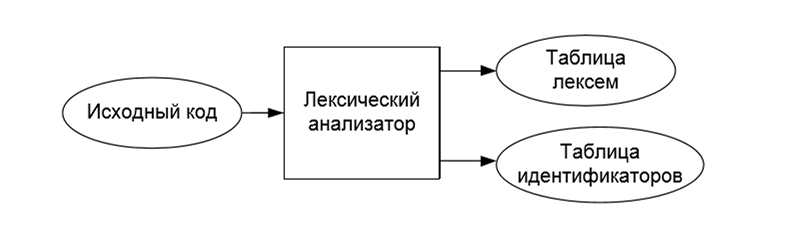


Рисунок 3.1 *–* Структура лексического анализатора BVS-2024

## **3.2. Контроль входных символов**

Таблица для контроля входных символов представлена в приложении Б.

Принцип работы таблицы: каждому элементу соответствует значение в шестнадцатеричной системе счисления.

В представленной таблице: SPACE – пробельные символы, F – запрещённый символы, T – разрешённый символы, I – игнорируемый символы.

## **3.3 Удаление избыточных символов**

Избыточный символ – это символ, отсутствие которого никоем образом не влияет на исходный текст программы.

В языке BVS-2024 символы пробела (0x20) и табуляции (0x09) являются избыточными: пробелы в начале строки и подряд идущие пробелы игнорируются, а табуляции полностью исключаются из текста. Они игнорируются при считывании из файла исходного кода на языке BVS-2024.

Описание алгоритма удаления избыточных символов:

* Посимвольно считывается файл с исходным кодом программы;
* В конце каждой итерации посимвольного считывания текущий символ сохраняется в буфер;
* встреча пробела включает проверку предыдущего символа;
* если текущий символ – кавычка, отключается игнорирование пробелов до тех пор, пока не встретится еще одна кавычка (т.к. внутри кавычек содержатся данные пользователя – литералы);

## **3.4 Перечень ключевых слов**

Все ключевые слова языка BVS-2024, символы операций, сепараторы, соответствующие им лексемы и регулярные выражения приведены в таблице 3.1.

Таблица 3.1 – Все ключевые слова, сепараторы и т.д. языка BVS-2024

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Описание |
| usint, text, boolean, symbol | t | Названия типов данных языка. |
| Идентификатор | i | Длина идентификатора. |
| Литерал | l | Литерал любого доступного типа. |
| function | f | Объявление функции. |
| if | y | Условный оператор |
| return | r | Выход из функции/процедуры. |
| main | m | Главная функция. |
| declare | d | Объявление переменной. |
| display | p | Оператор вывода. |
| : | : | Оператор присваивания. |
| ; | ; | Признак конца инструкции. |
| , | , | Разделитель параметров функции. |
| { | { | Начало блока функции. |
| } | } | Конец блока функции. |
| ( | ) | Открывающая скобка. |
| ) | ) | Закрывающая скобка. |
| ==, !=, <=, >= | k | Логические операторы. |

Граф перехода для логических литералов “true” и “false” представлен на рисунке 3.2. Фрагмент кода с реализацией данного графа н языке C++ представлен в листинге 3.1.

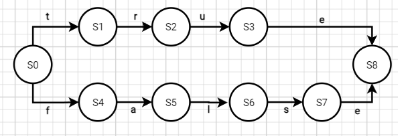


Рисунок 3.2 *–* Граф перехода конечного автомата для логических литерала

FST lex\_booleanLiteral(

str,

9,

NODE(2,

RELATION('t', 1),

RELATION('f', 4)

),

NODE(1, RELATION('r', 2)),

NODE(1, RELATION('u', 3)),

NODE(1, RELATION('e', 8)),

NODE(1, RELATION('a', 5)),

NODE(1, RELATION('l', 6)),

NODE(1, RELATION('s', 7)),

NODE(1, RELATION('e', 8)),

NODE()

);

Листинг 3.1 *–* Граф перехода конечного автомата для логических литерала

## **3.5 Основные структуры данных**

Таблицы лексем, токенов и идентификаторов – это важные структуры данных для лексического анализа и последующей обработки исходного кода. Каждая таблица хранит разные виды информации: токены для разбора синтаксиса, лексемы для сопоставления с грамматикой, и идентификаторы для хранения переменных и функций.

Использование этих таблиц позволяет эффективно управлять данными на разных этапах анализа программы, обеспечивая корректную работу компилятора или интерпретатора.

Структура лексемы и таблицы лексем представлена в листинге 3.2.

struct Entry {

int idxfirstLE; // индекс первой строки в таблице лексем

char id[ID\_MAXSIZE]{}; // имя идентификатора

IDDATATYPE iddatatype; // тип данных

IDTYPE idtype; // тип ид

union {

short vbyte; // значение byte

bool vboolean; // значение boolean

char vsymbol; // значение symbol

struct {

int len; // количество символов

char str[TI\_TEXT\_MAXSIZE - 1]; // символы

} vtext[TI\_TEXT\_MAXSIZE]; // значение text

} value;

struct LexTable {

int size; // текущий размер

std::vector<Entry>table; // строки таблицы лексем

};

Листинг 3.2 *–* Структура лексемы и таблицы лексем

Entry: Строка таблицы идентификаторов, которая содержит:

idxfirstLE – индекс первого вхождения идентификатора в таблице лексем,

id – строка, представляющая имя идентификатора,

iddatatype – тип данных идентификатора

value – структура, которая хранит значение идентификатора в зависимости от его типа.

LexTable: таблица идентификаторов, которая хранит множество структур: size – текущее количество записей в таблице идентификаторов, table – массив, содержащий все записи идентификаторов.

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

При возникновении ошибки в лексическом анализаторе формируется ошибка в следующем формате: номер ошибки, пояснительный текст, строка в исходном тексте, позиция в строке. Перечень сообщений лексического анализатора приведен в таблице 3.2.

Таблица 3.2 – Перечень сообщений лексического анализатора языка BVS-2024

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 118 | Идентификатор с таким же именем уже существует |
| 120 | Не удалось разобрать лексему из входного файла |
| 121 | Объявлен идентификатор без типа |
| 122 | Функция “main” объявлена более одного раза |
| 123 | Не объявлена функция “main” |
| 124 | Используется необъявленный идентификатор |
| 125 | Имя идентификатора превышает допустимое значение |

## **3.7 Принцип обработки ошибок**

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

## **3.8 Параметры лексического анализатора**

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

## **3.9 Алгоритм лексического анализа**

Лексический анализ является первой и наиболее простой фазой трансляции. Алгоритм лексического анализатора заключается в следующем:

* + - * сначала проверяется входной поток символов программы на исходном языке на допустимость, удаляются лишние пробелы;
      * после разбиения текста из файла с исходным кодом на слова, для каждого слова подбирается конечный автомат, способный его разобрать. В случае, если такой автомат существует, цепочка будет разобрана, иначе выводится сообщение об ошибке с указанием номера строки;
      * далее лексический анализатор анализирует лексему, соответствующую данному слову, и выполняет действия, описанные для данной лексемы;
      * формируется протокол работы.

Распознавание цепочек основывается на работе конечных автоматов. Работу конечного автомата можно проиллюстрировать с помощью графа переходов. Пример графа для цепочки “symbol” представлен на рисунке 3.3, где S0 – начальное, а S6 – конечное состояние автомата.

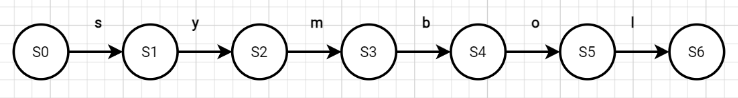


Рисунок 3.3 *–* Пример графа для цепочки “symbol”

## **3.10 Контрольный пример**

Результатом работы лексического анализатора в виде таблиц лексем представлен в приложении В и таблицы идентификаторов в приложении Г на основе исходного кода из приложения А.

# **4. Разработка синтаксического анализатора**

## **4.1 Структура синтаксического анализатора**

Синтаксический анализатор – часть транслятора, выполняющая синтаксический анализ. Входом для синтаксического анализа является таблица лексем и таблица идентификаторов. Выходом – дерево разбора.

В языке BVS-2024 синтаксический анализ выполняется после завершения работы лексического анализатора.

Структура синтаксического анализатора представлена на рисунке 4.1.



Рисунок 4.1 *–* Структура синтаксического анализатора

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

Синтаксис языка BVS-2024 описывается при помощи грамматики типа 2 иерархии Хомского (Контекстно-свободной грамматики).

Контекстно-свободная грамматика – грамматика типа 2 по иерархии Хомского. Данная грамматика имеет вид , где

* T – множество терминальных символов,
* N – множество нетерминальных символов,
* P – множество правил переходов,
* S – стартовый символ.
* В контекстно-свободной грамматике правила имеют вид:
* , где
* ,
* ,
* - словарь грамматики .

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

Таблица 4.1 – Описание правил, составляющих грамматику языка BVS-2024

|  |  |  |
| --- | --- | --- |
| Нетерминальный символ | Цепочки правил | Описание |
| S-> | dtfi(F){NR}S  dtfi(F){N}S  m{N} | Правила, порождающие главную функцию main и глобальные функции. |
| F-> | tiC  ,tiC | Правила, описывающие параметры функции. |
| C-> | ,tiF | Правила для дополнительной проверки параметров функции. |
| R-> | ri;  rl; | Правила для возврата значения функции. |
| N-> | dti;N  dti:E;N  i:E;N  p(O);N  i(p);N  y(W){N}N | Правила для тела функции или конструкции языка. |
| O-> | i  l | Правила для параметров специальной функции “display”. |
| E-> | i(P)  i  l | Правила для выражений. |
| P-> | iK  lK  ,iK  ,lK | Правила дляпараметров функции. |
| K-> | ,iP  ,lP | Вспомогательные правила для добавления нескольких параметров. |
| W-> | i(P)T  iT  lT | Правила для содержимого в скобках после “if”. |
| T-> | ki(P)  ki  kl | Вспомогательные правила для конструкций после “if”. |

## **4.3 Построение конечного магазинного автомата**

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

- множество состояний;

- алфавит входных символов;

- специальный алфавит магазинных символов;

-функция переходов автомата ,

где - множество подмножеств ;

- начальное состояние автомата;

- начальное состояние магазина (маркер дна);

- множество конечных состояний.

Конфигурация (текущее состояние автомата) описывается тройкой , где - текущее состояние автомата, - остаток цепочки, - цепочка-содержимое магазина.

Начальное состояние , - начальное состояние автомата, - входная цепочка, - маркер дна магазина.

Схема работы автомата с магазинной памятью представлена на рисунке 4.2.

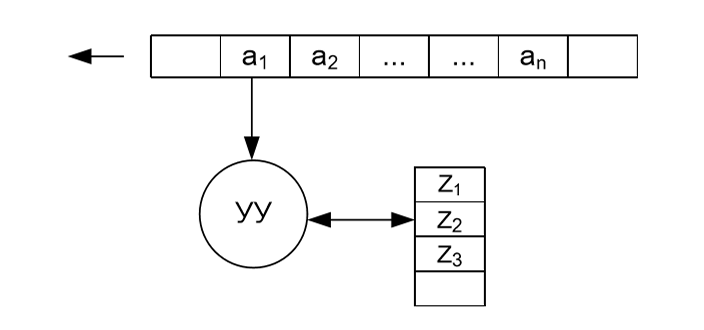


Рисунок 4.2 *–* Схема работы конечного автомата с магазинной памятью

Работа автомата

* состояние автомата
* читает символ находящийся под головкой (сдвигает ленту);
* не читает ничего (читает , не сдвигает ленту);
* из определяет новое состояние , если или .
* читает верхний (в стеке) символ и записывает цепочку т.к. , при этом, если , то верхний символ магазина просто удаляется.

Работа автомата заканчивается

## **4.4 Основные структуры данных**

Основные структуры данных синтаксического анализатора включают в себя структуру магазинного автомата и структуру грамматики Грейбах, описывающая правила языка BVS-2024. Структура синтаксического анализатора представлена в приложении Д.

## **4.5 Описание алгоритма синтаксического разбора**

Алгоритма работы синтаксического анализа изображен на рисунке 4.3. Его можно описать следующим образом:

* В магазин записывается стартовый символ;
* На основе полученных ранее таблиц формируется входная лента;
* Запускается автомат;
* Выбирается цепочка по первому символу, соответствующая нетерминальному, записывается в магазин в обратном порядке;
* Если терминалы в стеке и в ленте совпадают, то данный терминал удаляется из ленты и магазина. Иначе возвращаемся в предыдущее сохраненное состояние и выбираем другую цепочку нетерминала;
* Если в магазине встретился нетерминал, переходим к пункту 4;
* Если символ достиг дна стека, и лента в этот момент пуста, то синтаксический анализ выполнен успешно. Иначе генерируется исключение.

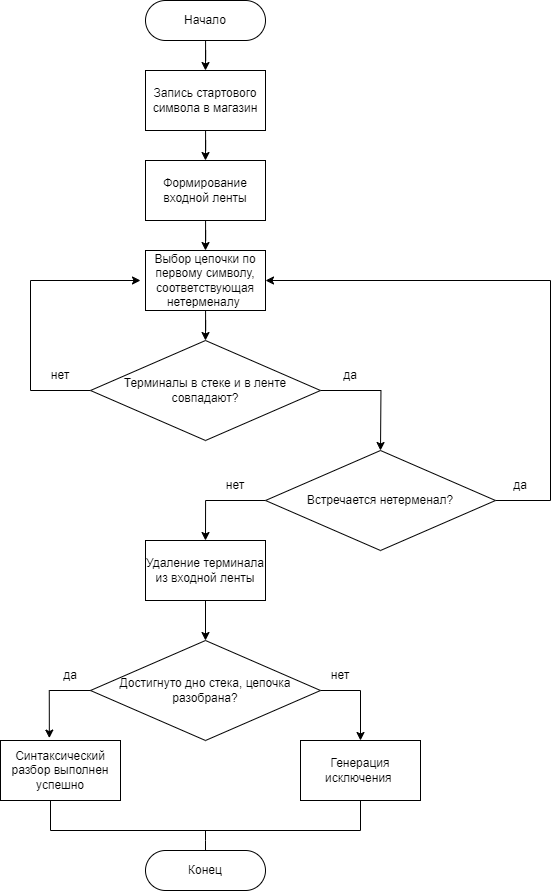


Рисунок 4.3 *–* Алгоритма работы синтаксического анализа

## **4.6 Структура и перечень сообщений синтаксического анализатора**

При возникновении ошибки в синтаксическом анализаторе формируется ошибка в следующем формате: Номер ошибки, пояснительный текст, строка в исходном тексте. Перечень сообщений синтаксического анализатора представлен в таблице 4.2.

Таблица 4.2 – Перечень сообщений синтаксического анализатора языка BVS-2024

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 600 | Неверная структура программы. |
| 601 | Ошибочный в операторе. |
| 602 | Ошибка в возвращаемом значении. |
| 603 | Ошибка в параметрах функции. |
| 604 | Ошибка в выражении. |

## **4.7 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.

## **4.8 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 4.5. Все ошибки записываются в общую структуру ошибок, а также отображаются на консоли.

## **4.9 Контрольный пример**

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

# **5. Разработка семантического анализатора**

## **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.

Схема семантического анализатора языка BVS-2024 представлена на рисунке 5.1.

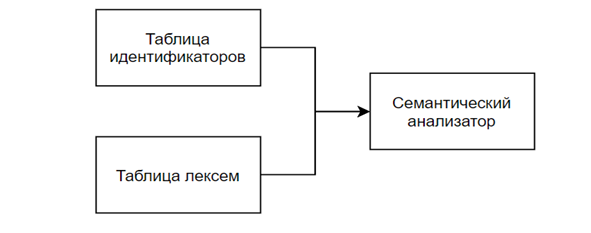


Рисунок 5.1 – Схема семантического анализатора языка BVS-2024

## **5.2 Функции семантического анализатора**

Семантические проверки языка BVS-2024 с указанием фаз их выполнения приведены в таблице 5.1.

Таблица 5.1 – Семантические проверки языка BVS-2024

|  |  |
| --- | --- |
| Семантическая проверка | Фаза выполнения |
| Повторная реализация функции “main” | Лексический анализ |
| Наличие точки входа в программу | Лексический анализ |
| Объявление идентификатора перед использованием | Семантический анализ |
| Повторное объявление идентификатора | Семантический анализ |
| Соответствие типов в выражении | Семантический анализ |
| Количество параметров функции | Семантический анализ |
| Соответствие типа возвращаемого значения типу функции | Семантический анализ |

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

Сообщения, формируемые семантическим анализатором в языке BVS-2024 представлены в таблице 5.2.

Таблица 5.2 – Перечень сообщений семантического анализатора языка BVS-2024

|  |  |
| --- | --- |
| Код ошибки | Сообщение |
| 700 | Тип возвращаемого значения функции не соответствует возвращаемому значению. |
| 701 | Не совпадает количество параметров функции при вызове и объявлении. |
| 702 | Не совпадают типы параметров функции при вызове и объявлении. |
| 703 | Типы данных, используемые в выражении, не совпадают. |
| 704 | Попытка вызвать операцию, неподдерживаемую типом данных. |
| 705 | Превышение максимального количества параметров функции (16). |
| 706 | Невозможно использовать идентификатор функции кроме как для её вызова. |
| 707 | Попытка присвоить идентификатору значение функции, которое отлично от типа данных идентификатора. |
| 710 | Конструкция if может принимать в качество проверки либо boolean, либо данные, которые совпадают по типу данных. |
| 711 | Display не принимает в качестве параметра функцию типа 'процедура'. |
| 714 | Display обязательно должно принимать какое-либо значение. |

## **5.4 Принцип обработки ошибок**

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

## **5.5 Контрольный пример**

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

# **6. Вычисление выражений**

## **6.1 Выражения, допускаемые языком**

В языке BVS-2024 допускаются выражения, применимые к целочисленным типам, допускается использование функций. Операции и их приоритетность приведены в таблице 6.1.

Таблица 6.1 – Операции и их приоритетность языка BVS-2024

|  |  |
| --- | --- |
| Приоритет | Операция |
| 0 | ( |
| 0 | ) |
| 1 | , |

Пример выражения из контрольного примера для выражения “prim: getNum(prim);”

prim prim getNum @1 :

* + - * prim – аргумент функции.
      * getNum – функция с 1 аргументом, помечается @1.
      * : – оператор присваивания.

## **6.2 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. Результат операции заменяет в выражении последовательность её операндов и символ операции. Результатом вычисления всего выражения является результат последней вычисленной операции.

Алгоритм преобразования выражений к 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.

## **6.3 Программная реализация обработки выражений**

Фрагмент кода, реализующего преобразование выражений в обратный польский формат представлен в листинге 6.1.

case LEX\_LEFTTHESIS:

{

left\_counter++;

stack.push(lexem);

break;

}

case LEX\_RIGHTTHESIS:

{

right\_counter++;

if (!ContainsElement(stack, stack\_size, LEX\_LEFTTHESIS))

return false;

while (stack.top() != LEX\_LEFTTHESIS) {

PolishString += stack.top();

stack.pop();

}

stack.pop();

if (!stack.empty() && stack.top() == '@') {

PolishString += stack.top() + toString(params\_counter - 1);

params\_counter = 0;

stack.pop();

}

break;

}

case LEX\_LITERAL: { if (std::find(stack.c.begin(), stack.c.begin(), '@') != stack.c.end()) params\_counter++;

PolishString += lexem;

if (lextable.table[i].idxTI != LT\_TI\_NULLIDX)

ids.push\_back(lextable.table[i].idxTI);

operands\_count++;

break;

}

Листинг 6.1 – Фрагмент кода, реализующего преобразование выражений

При встрече литерала, литерал помещается в выходную строку.

При встрече открывающей скобки, скобка кладется в стек.

При встрече закрывающей скобки, идет извлечение элементов из стека, пока не будет достигнута открывающая скобка.

## **6.4 Контрольный пример**

В приложении Ж представлена обратной польская запись для функций контрольного примера.

# **7. Генерация кода**

## **7.1 Структура генератора кода**

Генератор кода – часть транслятора, выполняющая генерацию ассемблерного кода на основе полученных данных на предыдущих этапах трансляции. На вход генератора подаются таблица идентификаторов, на основе которых генерируется файл с ассемблерным кодом.

Выходным будет являться файл, генерируемый во время компиляции, который будет содержать в себе транслированный исходный код.

Структура генератора кода BVS-2024 представлена на рисунке 7.1.

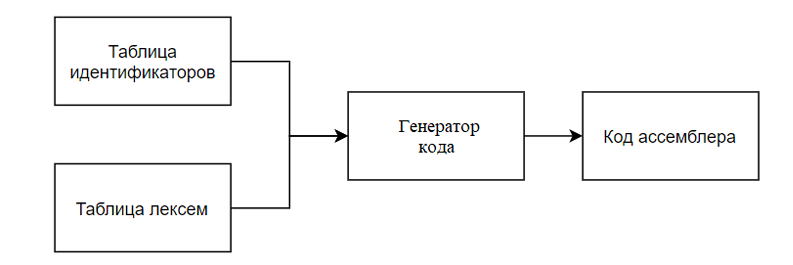


Рисунок 7.1 – Структура генератора кода

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

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

– .STACK – стек;

– .CONST – константы;

– .DATA – переменные;

– .CODE – код.

Соответствие типов данных в исходном языке программирования BVS-2024 типам целевого языка приведены в таблице 7.1.

Таблица 7.1 – Соответствие типов идентификаторов языка BVS-2024 и языка ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора языка BVS-2024 | Тип идентификатора ассемблера | Описание |
| byte | DWORD | Хранит беззнаковые целые числа |
| boolean | BYTE | 8-битное значение, 0(ложь), 1(истина) |
|  | | |
| Окончание таблицы 7.1 | | |
| text | DWORD | Хранит указатель на начало строки. |
| symbol | BYTE | 8-битное целое число, представляющее символ |

## **7.3 Статическая библиотека**

Функции стандартной библиотеки языка BVS-2024, представлены в таблице 1.9.

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

## **7.4 Особенности алгоритма генерации кода**

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

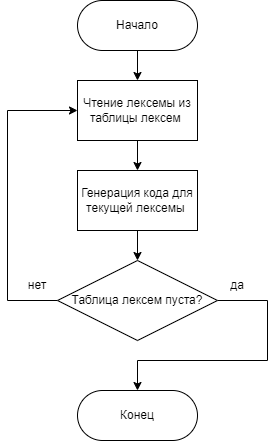


Рисунок 7.2 – Блок-схема алгоритма генерации кода языка ассемблера

Алгоритм генерации кода в данном примере используется для преобразования промежуточного представления программы (лексический и синтаксический анализ) в ассемблерный код. Основные этапы включают создание заголовка, констант, данных, обработку выражений и команд, таких как присваивание, вызов функции, условные операторы (например, if), возврат значений из функции и вывод данных.

## **7.5 Входные параметры, управляющие генерацией кода**

На вход генератору кода поступают таблицы лексем и идентификаторов. Результат работы генератора кода выводится в файл с расширением .asm.

Пример функции представлен в листинге 7.1.

void Functions(Out::OUT out, MFST::LEX lex) {

for (int i = 0; i < lex.idtable.size; i++) {

if (lex.idtable.table[i].idtype == IT::FUNCTION) {

\*out.stream << "\nF" << lex.idtable.table[i].id << " PROC uses ebx ecx edi esi";

int cur = 1;

while (lex.lextable.table[lex.idtable.table[i].idxfirstLE + cur].lexema != LEX\_RIGHTTHESIS) {

// Обработка параметров функции

if (lex.lextable.table[lex.idtable.table[i].idxfirstLE + cur].lexema == LEX\_ID) {

\*out.stream << ", " << lex.idtable.table[lex.lextable.table[lex.idtable.table[i].idxfirstLE + cur].idxTI].id;

}

cur++;

}

int startPos = lex.idtable.table[i].idxfirstLE + cur;

while (lex.lextable.table[lex.idtable.table[i].idxfirstLE + cur].lexema != LEX\_RETURN) {

cur++;

}

int endPos = lex.idtable.table[i].idxfirstLE + cur;

Expression(out, lex, startPos, endPos); // Генерация кода тела функции

\*out.stream << "ret\n";

\*out.stream << "F" << lex.idtable.table[i].id << " ENDP\n";

}

}

}

Листинг 7.1 – Пример генерации функции в код ассемблера

Head: генерирует заголовок программы, включая подключения библиотек и объявление прототипов функций.

Const: генерирует секцию .const, где определяются все константы программы.

Data: генерирует секцию .data, где объявляются переменные программы.

Expression: генерирует код для обработки выражений, таких как присваивания и вызовы функций. Для каждой лексемы проверяются типы данных, и в зависимости от этого генерируются соответствующие ассемблерные команды.

Functions: генерирует код для обработки функций, включая их прототипы и тела. Внутри этой функции также генерируются вызовы других функций с передачей параметров и обработка возвращаемых значений.

В этой функции генерируется код для каждой функции, включая параметры и тело функции, и добавляются инструкции для возврата значений.

## **7.6 Контрольный пример**

Результат генерации кода на основе контрольного примера из приложения А представлен в приложении И. На рисунке 7.3 приведен результат работы контрольного примера.

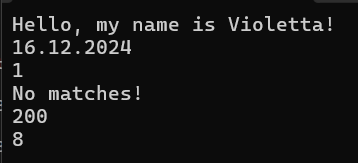


Рисунок 7.3 – Результат работы программы на языке BVS-2024

# **8. Тестирование транслятора**

## **8.1 Общие положения**

В результате обработки исходного кода программы, представленного в приложении А, транслятор языка BVS-2024 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.

## **8.1 Результаты тестирования**

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

Таблица 8.1– Описание тестовых наборов языка BVS-2024

|  |  |
| --- | --- |
| Исходный код с ошибкой | Генерируемое сообщение об ошибке |
| declare usint #number: 120; | Ошибка 111 :IN: Недопустимый символ в исходном файле |
| declare symbo sym; | Ошибка 124 :FST: Используется необъявленный идентификатор |
| declare uisnt number: 120;  declare usint number; | Ошибка 128 :FST: Идентификатор используется дважды |
| declare number: 120; | Ошибка 121 :FST: Объявлен идентификатор без типа |
| declare usint function getNum(uisnt n) {return 'n';} | Ошибка 700 :SA: Тип возвращаемого значения функции не соответствует возвращаемому значению |
| declare usint function checkName(text inname) {  declare usint res;  res:StrCmp(inname,"Mike0")  return res;}  number: checkName(txt, txt); | Ошибка 702 :SA: Не совпадают типы параметров функции при вызове и объявлении |
| main {main{}} | Ошибка 122 :FST: Функция main объявлена более одного раза |
| display(); | Ошибка 714 :SA: Display обязательно должно принимать какое-либо значение |
| declare usint function getNum(usint n, usint b) {return n;}prim:getNum(prim); | Ошибка 701 :SA: Не совпадает количество параметров функции при вызове и объявлении |

# **Заключение**

В ходе выполнения курсовой работы был разработан транслятор для языка программирования BVS-2024. Таким образом, были выполнены основные задачи данной курсовой работы:

* сформулирована спецификация языка BVS-2024;
* разработаны конечные автоматы и алгоритмы для реализация лексического анализатора;
* разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
* разработан семантический анализатор, осуществляющий проверку смысла используемых инструкций;
* разработан транслятор с языка программирования BVS-2024 на язык ассемблера;
* проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка BVS-2024 включает:

* 5 типов данных;
* Поддержка операции вывода;
* Возможность вызова функций стандартной библиотеки;
* Наличие 6 логических операторов;
* Наличие условного оператора;
* Структурированная система для обработки ошибок пользователя.

# **Список использованных источников**

1. Курс лекций по КПО / Наркевич А.С
2. Прата, С. Язык программирования С++. Лекции и упражнения / С. Прата. – М., 2006 – 1104 c.
3. Ирвин К. Р. Язык ассемблера для процессоров Intel / К. Р. Ирвин. – M.: Вильямс, 2005. – 912с.
4. Ахо, А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003. – 768с.
5. Страуструп, Б. Принципы и практика использования C++ / Б. Страуструп – 2009 – 1238 с
6. Польская запись [Электронный ресурс] – Режим доступа: <https://studfiles.net/preview/2792990/>

# **Приложение А**

Контрольный пример

declare boolean function setBool(usint a) {

display(1);

return true;

}

declare hallow function sayHello() {

display("Hello, my name is Violetta!");

declare text todaysData: "16.12.2024";

display(todaysData);

}

declare hallow function getSetBool() {

setBool(1);

}

main {

sayHello();

getSetBool();

if (-1 == StrCmp("18", "1")) {

display("No matches!");

}

declare text str : "";

declare usint num;

num: 200;

display(num);

num: Pow(2, 3);

display(num);}

Листинг 1 - Контрольный пример

Контрольный пример, содержащий 3 семантические ошибки

declare boolean function setBool(usint a) {

display(1);

return 23;

}

declare hallow function sayHello() {

display("Hello, my name is Violetta!");

declare text todaysData: "16.12.2024";

display(todaysData);

}

declare hallow function getSetBool() {

setBool(1);

}

main {

sayHello();

getSetBool();

if (-1 == StrCmp("18", "1")) {

display("No matches!");

}

declare text str : "";

declare usint num;

num: 200;

display(num);

num: Pow(2, “txt”);

display();}

Листинг 2 - Контрольный пример, содержащий 3 семантические ошибки

Допущенные семантические ошибки:

- Ошибка 700 :SA: Тип возвращаемого значения функции не соответствует возвращаемому значению, строка 3, позиция 1

- Ошибка 713 :SA: Функции стандартной библиотеки должны принимать два параметра типа данных usint или два параметра типа данных text для StrCmp, строка 27, позиция 3

- Ошибка 714 :SA: Display обязательно должно принимать какое-либо значение, строка 28, позиция 3

# **Приложение Б**

Листинг – Таблица допустимых и запрещенных символов

#define IN\_CODE\_TABLE {\

IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::SPACE, IN::T, IN::F, IN::F, IN::F, IN::F, IN::F,\

IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,\

IN::SPACE, IN::T, IN::T, IN::F, IN::F, IN::F, IN::F, IN::T, IN::T, IN::T, IN::F, IN::F, IN::T, IN::T, IN::T, IN::F,\

IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,\

IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,\

IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::F, IN::T,\

IN::F, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T,\

IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::T, IN::F, IN::T, IN::F, IN::F,\

\

IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,\

IN::F, IN::T, IN::T, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,\

IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,\

IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,\

IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,\

IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,\

IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F, IN::F,\

}

# **Приложение В**

Листинг – Таблица лексем

Позиция Строка Столбец Лексема ТИ

0 1 1 d -1

1 1 2 t -1

2 1 3 f -1

3 1 4 i 3

4 1 5 ( -1

5 1 6 t -1

6 1 7 i 4

7 1 8 ) -1

8 1 9 { -1

9 2 1 p -1

10 2 2 ( -1

11 2 3 l 5

12 2 4 ) -1

13 2 5 ; -1

14 3 1 r -1

15 3 2 l 6

16 3 3 ; -1

17 4 1 } -1

18 6 1 d -1

19 6 2 t -1

20 6 3 f -1

21 6 4 i 7

22 6 5 ( -1

23 6 6 ) -1

24 6 7 { -1

25 7 1 p -1

26 7 2 ( -1

27 7 3 l 8

28 7 4 ) -1

29 7 5 ; -1

30 8 1 d -1

31 8 2 t -1

32 8 3 i 9

33 8 4 : -1

34 8 5 l 10

35 8 6 ; -1

36 9 1 p -1

37 9 2 ( -1

38 9 3 i 9

39 9 4 ) -1

40 9 5 ; -1

41 10 1 } -1

42 12 1 d -1

43 12 2 t -1

44 12 3 f -1

45 12 4 i 11

46 12 5 ( -1

47 12 6 ) -1

48 12 7 { -1

49 13 1 i 3

50 13 2 ( -1

51 13 3 l 12

52 13 4 ) -1

53 13 5 ; -1

54 14 1 } -1

55 17 1 m -1

56 17 2 { -1

57 18 1 i 7

58 18 2 ( -1

59 18 3 ) -1

60 18 4 ; -1

61 19 1 i 11

62 19 2 ( -1

63 19 3 ) -1

64 19 4 ; -1

65 20 1 y -1

66 20 2 ( -1

67 20 3 l 13

68 20 4 k -1

69 20 5 i 2

70 20 6 ( -1

71 20 7 l 14

72 20 8 , -1

73 20 9 l 15

74 20 10 ) -1

75 20 11 ) -1

76 20 12 { -1

77 21 1 p -1

78 21 2 ( -1

79 21 3 l 16

80 21 4 ) -1

81 21 5 ; -1

82 22 1 } -1

83 23 1 d -1

84 23 2 t -1

85 23 3 i 17

86 23 4 : -1

87 23 5 l 18

88 23 6 ; -1

89 24 1 d -1

90 24 2 t -1

91 24 3 i 19

92 24 4 ; -1

93 25 1 i 19

94 25 2 : -1

95 25 3 l 20

96 25 4 ; -1

97 26 1 p -1

98 26 2 ( -1

99 26 3 i 19

100 26 4 ) -1

101 26 5 ; -1

102 27 1 i 19

103 27 2 : -1

104 27 3 i 0

105 27 4 ( -1

106 27 5 l 21

107 27 6 , -1

108 27 7 l 22

109 27 8 ) -1

110 27 9 ; -1

111 28 1 p -1

112 28 2 ( -1

113 28 3 i 19

114 28 4 ) -1

115 28 5 ; -1

116 30 1 } -1

# **Приложение Г**

Листинг – Таблица идентификаторов

Pow : Первое вхождение в таблицу лексем: 0 . Тип: литерал . Тип данных: целочисленный

Sum : Первое вхождение в таблицу лексем: 1 . Тип: литерал . Тип данных: целочисленный

StrCmp : Первое вхождение в таблицу лексем: 2 . Тип: литерал . Тип данных: целочисленный

setBool : Первое вхождение в таблицу лексем: 3 . Тип: функция . Тип данных: логический

setBool$a : Первое вхождение в таблицу лексем: 6 . Тип: параметр . Тип данных: целочисленный

LIT1 : Первое вхождение в таблицу лексем: 11 . Тип: литерал . Тип данных: целочисленный

LIT2 : Первое вхождение в таблицу лексем: 15 . Тип: литерал . Тип данных: логический

sayHello : Первое вхождение в таблицу лексем: 21 . Тип: функция . Тип данных: процедура

LIT3 : Первое вхождение в таблицу лексем: 27 . Тип: литерал . Тип данных: строчный

sayHello$todaysData : Первое вхождение в таблицу лексем: 32 . Тип: переменная. Тип данных: строчный

LIT4 : Первое вхождение в таблицу лексем: 34 . Тип: литерал . Тип данных: строчный

getSetBool : Первое вхождение в таблицу лексем: 45 . Тип: функция . Тип данных: процедура

LIT5 : Первое вхождение в таблицу лексем: 51 . Тип: литерал . Тип данных: целочисленный

LIT6 : Первое вхождение в таблицу лексем: 67 . Тип: литерал . Тип данных: целочисленный

LIT7 : Первое вхождение в таблицу лексем: 71 . Тип: литерал . Тип данных: строчный

LIT8 : Первое вхождение в таблицу лексем: 73 . Тип: литерал . Тип данных: строчный

LIT9 : Первое вхождение в таблицу лексем: 79 . Тип: литерал . Тип данных: строчный

main$str : Первое вхождение в таблицу лексем: 85 . Тип: переменная. Тип данных: строчный

LIT10 : Первое вхождение в таблицу лексем: 87 . Тип: литерал . Тип данных: строчный

main$num : Первое вхождение в таблицу лексем: 91 . Тип: переменная. Тип данных: целочисленный

LIT11 : Первое вхождение в таблицу лексем: 95 . Тип: литерал . Тип данных: целочисленный

LIT12 : Первое вхождение в таблицу лексем: 106. Тип: литерал . Тип данных: целочисленный

LIT13 : Первое вхождение в таблицу лексем: 108. Тип: литерал . Тип данных: целочисленный

# **Приложение Д**

Листинг – Структур данных, описывающие контекстно-свободную грамматику

struct Rule // правило в грамматике Грейбах

{

GRBALPHABET nn; // нетерминал

int idError;

short size;

struct Chain

{

short size;

GRBALPHABET\* nt;

Chain() { size = 0; nt = new GRBALPHABET[1]{}; };

Chain(

short psize,

GRBALPHABET s, ...

);

char\* getCChain(char\* b);

static GRBALPHABET T(char t) { return GRBALPHABET(t); }; // терминал

static GRBALPHABET N(char n) { return -GRBALPHABET(n); }; // не терминал

static bool isT(GRBALPHABET s) { return s > 0; };

static bool isN(GRBALPHABET s) { return !isT(s); };

static char alphabet\_to\_char(GRBALPHABET s) { return isT(s) ? char(s) : char(-s); }; // GRBALPHABET->char

}\*chains;

Rule() { nn = 0x00, size = 0; };

Rule(

GRBALPHABET pnn,

int iderror,

short psize,

Chain c, ...

);

char\* getCRule(

char\* b, // буфер

short nchain

);

short getNextChain(

GRBALPHABET t, // первый символ цепочки

Chain& pchain, // возвращаемая цепочка

short j // номер цепочки

);

};

struct Greibach // грамматика Грейбах

{

short size;

GRBALPHABET startN;

GRBALPHABET stbottomT;

Rule\* rules;

Greibach() { size = 0; startN = 0; stbottomT = 0; rules = 0; };

Greibach(

GRBALPHABET pstartN,

GRBALPHABET pstbottomT,

short psize,

Rule r, ...

);

short getRule(

GRBALPHABET pnn,

Rule& prule

);

Rule getRule(short n);

};

Greibach getGreibach();

# **Приложение Е**

Листинг – Последовательность правил грамматики

0 : S->dtfi(F){NR}S

5 : F->tiC

7 : C->

9 : N->p(O);N

11 : O->l

14 : N->

14 : R->rl;

18 : S->dtfi(F){N}S

23 : F->

25 : N->p(O);N

27 : O->l

30 : N->dti:E;N

34 : E->l

36 : N->p(O);N

38 : O->i

41 : N->

42 : S->dtfi(F){N}S

47 : F->

49 : N->i(P);N

51 : P->lK

52 : K->

54 : N->

55 : S->m{N}

57 : N->i(P);N

59 : P->

61 : N->i(P);N

63 : P->

65 : N->y(W){N}N

67 : W->lT

68 : T->ki(P)

71 : P->lK

72 : K->,lP

74 : P->

77 : N->p(O);N

79 : O->l

82 : N->

83 : N->dti:E;N

87 : E->l

89 : N->dti;N

93 : N->i:E;N

95 : E->l

97 : N->p(O);N

99 : O->i

102 : N->i:E;N

104 : E->i(P)

106 : P->lK

107 : K->,lP

109 : P->

111 : N->p(O);N

113 : O->i

# **Приложение Ж**

Листинг – обратная польская запись для функций контрольного примера

il@1

i@0

i@0

ill@2

ill@2

Польская запись выполнена без ошибок

dtfi(ti){p(l);rl;}dtfi(){p(l);dti:l;p(i);}dtfi(){il@1;}m{i@0;i@0;y(lkill@2#){p(l);}dti:l;dti;i:l;p(i);i:ill@2#;p(i);})))

# **Приложение И**

Листинг – Результат генерации кода

.586P

.model flat, stdcall

includelib libucrt.lib

includelib kernel32.lib

includelib ../Debug/BVS-2024Lib.lib

ExitProcess PROTO : DWORD

SetConsoleCP PROTO : DWORD

SetConsoleOutputCP PROTO : DWORD

Pow PROTO : BYTE, : BYTE

Sum PROTO : BYTE, : BYTE

StrCmp PROTO : DWORD, : DWORD

writestr PROTO : DWORD

writeint PROTO : BYTE

writebool PROTO : BYTE

writechar PROTO : BYTE

.stack 4096

.const

LIT1 DWORD 1 ; byte

LIT2 BYTE 1 ; boolean

LIT3 DB "Hello, my name is Violetta!", 0 ; text

LIT4 DB "16.12.2024", 0 ; text

LIT5 DWORD 1 ; byte

LIT6 DWORD 4294967295 ; byte

LIT7 DB "18", 0 ; text

LIT8 DB "1", 0 ; text

LIT9 DB "No matches!", 0 ; text

LIT10 DB 0, 0 ; text

LIT11 DWORD 200 ; byte

LIT12 DWORD 2 ; byte

LIT13 DWORD 3 ; byte

.data

sayHello$todaysData DWORD 0 ; text

main$str DWORD 0 ; text

main$num DWORD 0 ; byte

.code

FsetBool PROC uses ebx ecx edi esi, setBool$a : DWORD

mov eax, LIT1

push eax

CALL writeint

;return

movzx eax, LIT2

ret

FsetBool ENDP

FsayHello PROC uses ebx ecx edi esi

push offset LIT3

CALL writestr

; string #8 : i:l

push offset LIT4

pop eax

mov sayHello$todaysData, eax

push sayHello$todaysData

CALL writestr

ret

FsayHello ENDP

FgetSetBool PROC uses ebx ecx edi esi

mov eax, LIT5

push eax

CALL FsetBool

ret

FgetSetBool ENDP

main PROC

Invoke SetConsoleCP, 1251

Invoke SetConsoleOutputCP, 1251

CALL FsayHello

CALL FgetSetBool

If\_Begin1:

mov eax, LIT6

push eax

push OFFSET LIT7

push OFFSET LIT8

CALL StrCmp

push eax

pop ebx

pop eax

cmp eax, ebx

je If\_End1

jmp If\_End2

If\_End1:

push offset LIT9

CALL writestr

If\_End2:

; string #23 : i:l

push offset LIT10

pop eax

mov main$str, eax

; string #25 : i:l

push LIT11

pop eax

mov main$num, eax

mov eax, main$num

push eax

CALL writeint

; string #27 : i:ill@2#

push LIT13

push LIT12

CALL Pow

mov main$num, eax

mov eax, main$num

push eaxCALL writeint

push -1call ExitProcess

main ENDPend main