Содержание

Введение 5

[Глава 1 Спецификация языка программирования 6](#_Toc91196429)

[1.1 Характеристика языка программирования 6](#_Toc91196430)

[1.2 Определение алфавита языка программирования 6](#_Toc91196431)

[1.3 Применяемые сепараторы 6](#_Toc91196432)

[1.4 Применяемые кодировки 7](#_Toc91196433)

[1.5 Типы данных 8](#_Toc91196434)

[1.6 Преобразование типов данных 9](#_Toc91196435)

[1.7 Идентификаторы 9](#_Toc91196436)

[1.8 Литералы 9](#_Toc91196437)

[1.9 Объявления данных 10](#_Toc91196438)

[1.10 Инициализация данных 10](#_Toc91196439)

[1.11 Инструкции языка 10](#_Toc91196440)

[1.12 Операции языка 11](#_Toc91196441)

[1.13 Выражения и их вычисления 11](#_Toc91196442)

[1.14 Конструкции языка 11](#_Toc91196457)

[1.15 Область видимости идентификаторов 12](#_Toc91196458)

[1.16 Семантические проверки 12](#_Toc91196459)

[1.17 Распределение оперативной памяти на этапе выполнения 13](#_Toc91196460)

[1.18 Стандартная библиотека и её состав 13](#_Toc91196461)

[1.19 Ввод и вывод данных 13](#_Toc91196462)

[1.20 Точка входа 14](#_Toc91196463)

[1.21 Препроцессор 14](#_Toc91196464)

[1.22 Соглашения о вызовах 14](#_Toc91196465)

[1.23 Объектный код 14](#_Toc91196466)

[1.24 Классификация сообщений транслятора 14](#_Toc91196467)

[1.25 Контрольный пример 14](#_Toc91196468)

[Глава 2 Структура транслятора 15](#_Toc91196469)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 15](#_Toc91196470)

[2.2 Перечень параметров транслятора 16](#_Toc91196471)

[2.3 Протоколы, формируемые транслятором 16](#_Toc91196472)

[Глава 3 Разработка лексического анализатора 17](#_Toc91196473)

[3.1 Структура лексического анализатора 17](#_Toc91196474)

[3.2 Входные и выходные данные лексического анализатора 17](#_Toc91196475)

[3.3 Параметры лексического анализатора 18](#_Toc91196476)

[3.4. Алгоритм лексического анализатора 19](#_Toc91196477)

[3.5 Контроль входных символов 19](#_Toc91196478)

[3.6 Удаление избыточных символов 20](#_Toc91196479)

[3.7 Перечень ключевых слов 20](#_Toc91196480)

[3.8 Основные структуры данных 21](#_Toc91196481)

[3.9 Структура и перечень сообщений лексического анализатора 21](#_Toc91196482)

[3.10 Принцип обработки ошибок 22](#_Toc91196483)

[3.11 Контрольный пример 22](#_Toc91196483)

[Глава 4 Разработка синтаксического анализатора 23](#_Toc91196484)

[4.1 Структура синтаксического анализатора. 23](#_Toc91196485)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 23](#_Toc91196486)

[4.3 Построение конченого магазинного автомата 24](#_Toc91196487)

[4.4 Основные структуры данных 25](#_Toc91196488)

[4.5 Описание алгоритма синтаксического разбора 25](#_Toc91196489)

[4.6 Параметры синтаксического анализатора и режимы его работы 26](#_Toc91196491)

[4.7 Структура и перечень сообщений синтаксического анализатора 2](#_Toc91196491)6

[4.8 Принцип обработки ошибок 2](#_Toc91196492)6

[4.9 Контрольный пример 27](#_Toc91196493)

[Глава 5 Разработка семантического анализатора 28](#_Toc91196494)

[5.1 Структура семантического анализатора 28](#_Toc91196496)

[5.2 Функции семантического анализатора 28](#_Toc91196497)

[5.3 Структура и перечень сообщений семантического анализатора 28](#_Toc91196498)

[5.4 Принцип обработки ошибок 29](#_Toc91196499)

[5.5 Контрольный пример 29](#_Toc91196500)

[Глава 6 Преобразование выражений 30](#_Toc91196501)

[6.1 Выражения, допускаемые языком 30](#_Toc91196502)

[6.2 Польская запись и принцип ее построения 30](#_Toc91196503)

[6.3 Программная реализация обработки выражений 31](#_Toc91196504)

[6.4 Контрольный пример 31](#_Toc91196505)

[Глава 7 Генерация кода 32](#_Toc91196506)

[7.1 Структура генератора кода 32](#_Toc91196507)

[7.2 Представление типов данных в оперативной памяти 32](#_Toc91196508)

[7.3 Статическая библиотека 33](#_Toc91196509)

[7.4 Особенности алгоритма генерации кода 33](#_Toc91196510)

[7.5 Входные параметры, управляющие генерацией кода 34](#_Toc91196511)

[7.5 Контрольный пример 34](#_Toc91196512)

[Глава 8 Тестирование транслятора 35](#_Toc91196506)

[8.1 Общие положения 35](#_Toc91196513)

[8.2 Результаты тестирования 35](#_Toc91196513)

[Заключение 37](#_Toc91196514)

[Список используемых источников 38](#_Toc91196514)

[Приложение А 39](#_Toc91196515)

[Приложение Б 40](#_Toc91196516)

[Приложение В 43](#_Toc91196517)

[Приложение Г 46](#_Toc91196518)

[Приложение Д 49](#_Toc91196519)

[Приложение Е 51](#_Toc91196520)

# **Введение**

Целью курсового проекта является создание своего языка программирования и компилятора для его обработки. Язык был назван CYR-2023.

Компилятор – это программа, которая преобразует исходный код программы, написанный на одном языке программирования (называемом исходным языком), в машинный код или код на другом языке программирования (называемом целевым языком). Основная задача компилятора заключается в том, чтобы перевести исходный код программы в форму, понятную компьютеру, таким образом, чтобы программа могла быть выполнена. Компилятор CYR-2023 состоит из следующих частей:

* лексический анализатор;
* синтаксический анализатор;
* семантический анализатор;
* генератор исходного кода на языке ассемблера.

В соответсвии с курсовым проектом были поставлены следующие задачи:

– разработка спецификации языка программирования;

– разработка структуры транслятора;

– разработка лексического анализатора;

– разработка синтаксического анализатора;

– разработка семантического анализатора;

– разработка преобразования выражений;

– разработка генератора кода на язык ассемблер;

– тестирование.

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

**1. Спецификация языка программирования**

**1.1 Характеристика языка программирования**

Язык программирования CYR-2023 является универсальным языком высокого уровня. Он является компилируемым, транслируемым. Язык строго типизируемый, что говорит о невозможности преобразования типов.

**1.2 Определение алфавита языка программирования**

Совокупность символов, применяемых в определенном языке, определяется как алфавит этого языка. Структура алфавита языка CYR-2023 описана формально в виде набора правил в формате Бэкуса-Наура, представленных в таблице 1.1.

Таблица 1.1 – Алфавит языка программирования CYR-2023

|  |
| --- |
| <строчная буква латинского алфавита>::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z |
| <прописная буква латинского алфавита>::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z |
| <цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| <строчная буква русского алфавита>::= а|б|в|г|д|е|ё|ж|з|и|й|к|л|м|н|о|п|р|с|т|у|ф|х|ц|ч|ш|щ|ъ|ь|э|ю|я |
| <прописная буква русского алфавита>::= А|Б|В|Г|Д|Е|Ё|Ж|З|И|Й|К|Л|М|Н|О|П|Р|С|Т|У|Ф|Х|Ц|Ч|Ш|Щ|Ъ|Ь|Э|Ю|Я |
| <арифметическая операция>::= +|-|\*|/|[\] |
| <символ- сепаратор>::= ' '|,|(|)|{|}|; |
| <логическая операция>::= >|<|!= |

Исходный код CYR-2023 может содержать символы латинского алфавита, цифры десятичной системы счисления (от 0 до 9), символы арифметических операций и операций сравнения, а также символы-разделители. Русские символы разрешены только в строковых литералах.

**1.3 Применяемые сепараторы**

Перечень используемых сепараторов представлен в таблице 1.2.

Таблица 1.2 - Применяемые сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение |
| ; | Разделение инструкций |

Окончание таблицы 1.2 – Применяемые сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение |
| , (запятая) | Разделение параметров функции |
| = | Оператор присвоения |
| >, <, != | Символы сравнения |
| +, -, \*, / | Арифметические операции |
| {  } | Программный блок инструкций |
| (  ) | Параметры функций/приоритетность операций (в выражениях) |
| пробел | Разделитель цепочек. Допускается везде, кроме названий идентификаторов и ключевых слов |
| [, ] | Операторы сдвигов (влево и вправо соответственно) |

При обработке исходного текста программы, написанной на языке программирования, символы-разделители используются для разделения последовательностей языка, обеспечивая структурирование исходного текста.

**1.4 Применяемые кодировки**

Символы, поддерживающиеся в кодировке Windows-1251 представлены на рисунке 1.1.



Рисунок 1.1 – Таблица кодировки Windows-1251

Для написания исходного текста программ на языке CYR-2023 используется кодировка Windows-1251.

**1.5 Типы данных**

Допускается использование фундаментальных типов данных определенных в таблице 1.3.

Таблица 1.3 - Фундаментальные типы данных

|  |  |
| --- | --- |
| Тип данных | Описание |
| integer | Является целочисленным типом данных.  Этот тип данных занимает 4 байта.  Предусмотрен для объявления целочисленных данных.  Инициализация по умолчанию: 0.  Предусмотрены следующие арифметические операции:  + - бинарная операция суммирования(num+num);  - - бинарная операция вычитания(num-num);  \* - бинарная операция умножения(num\*num);  / - бинарная операция деления(num/num).  [ и ] – сдвиговые операции (для двоичной системы счисления) |
| string | Является строковым типом данных.  Предназначен для работы с несколькими символами.  Каждый символ занимает 1 байт.  Максимальное количество символов – 70, включая символ окончания строки. Минимальное количество – 2.  Инициализация по умолчанию: нулевой символ. |
| unsigned integer | Является беззнаковым целочисленным типом данных.  Этот тип данных занимает 4 байта.  Предназначен для объявления целочисленных положительных данных.  Инициализация по умолчанию: 0.  Предусмотрены следующие арифметические операции:  + - бинарная операция суммирования(num+num);  - - бинарная операция вычитания(num-num);  \* - бинарная операция умножения(num\*num);  / - бинарная операция деления(num/num).  [ и ] – сдвиговые операции (для двоичной системы счисления) |
| char | Является символьным типом данных.  Предназначен для работы с символами.  Каждый символ занимает 1 байт. |

Пользовательские типы данных не поддерживаются.

**1.6 Преобразование типов данных**

Предусмотрено преобразования из двоичной и восьмеричной систем счисления в десятичную.

**1.7 Идентификаторы**

### Для именования функций, параметров и переменных используются идентификаторы.

### Имена идентификаторов могут содержать символы латинского алфавита нижнего регистра и цифры. Имя идентификатора должно начинать с латинской буквы и не должно совпадать с ключевыми словами.

### Максимальная длина имени идентификатора равна десяти символам.

**1.8 Литералы**

Предусмотрены числовые, символьные, строковые и логические литералы. Правила записи приведены в таблице 1.4.

Таблица 1.4 - Правила записи литералов

|  |  |
| --- | --- |
| Тип литерала | Описание литерала |
| Числовые | Могут состоять только из цифр без дробной части.  Могут состоять только из цифр [0..9] без дробной части при выводе.  При выходе за пределы допустимости будет выведена соответствующая ошибка.  Двоичная система счисления: 2x[01]\*  Восьмеричная система счисления: [0-7]\* |
| Символьные | Состоит из 1-го символа, заключенного в ' ' (одинарные кавычки).  В случае превышения длины литерала работа транслятора прекращается. |
| Строковые | Состоит из символов, заключенных в ' ' (одинарные кавычки).  В случае превышения длины литерала работа транслятора прекращается. |
| Логические | Логический литерал представляет собой значение булевского типа данных, которое может быть либо истинным (true), либо ложным (false), которые преобразуются в 1 и 0 соответственно. |

Литерал – это неизменное значение, такое как константа или фиксированное выражение (например, число или строка), прямо включенное в программу.

**1.9 Объявление данных**

В языке программирования CYR-2023 переменная должна быть объявлена до ее использования. Областью видимости переменной является блок функции, в которой она определена. Не допустимо объявление глобальных переменных. Область видимости сверху вниз.

**1.10 Инициализация данных**

Инициализация данных подразумевает под собой присвоение переменной значения в месте её объявления. При объявлении переменной допускается инициализация данных.

При этом переменной будет присвоено значение литерала или идентификатора, стоящего справа от знака равенства. Объектами-инициализаторами могут быть идентификаторы, литералы, выражения и вызовы функций.

Предусмотрены значения по умолчанию, если переменные не инициализированы: значение 0 для целочисленного и беззнакового целочисленного типов данных и строка ‘’ (в консоль выведет "Empty line!") для строкового типа данных.

**1.11 Инструкции языка**

Инструкции языка программирования CYR-2023 представлены ниже в таблице 1.5.

Таблица 1.5 - Инструкции языка

|  |  |
| --- | --- |
| Инструкция языка | Синтаксис |
| Объявление функции | <declare> <тип данных> function <идентификатор> (<тип данных> <идентификатор>, …)  {  …  } |
| Объявление переменной | <declare><тип данных><идентификатор> |
| Присваивание | <идентификатор> = <литерал>;  <идентификатор> = <выражение>;  <идентификатор 1> = <идентификатор 2>; |
| Вывод данных | write<идентификатор / литерал> |
| Условный оператор | If(<выражение>;){…} |

Инструкции в языке программирования представляют собой отдельные команды или операции, осуществляющие определенные действия. Эти элементы являются фундаментальными строительными блоками программы, регулирующими ход выполнения кода.

**1.12 Операции языка**

Все операции в языке CYR-2023 применимы только к целочисленным типам данных.

Операции в языке программирования CYR-2023 приведены в таблице 1.6.

Таблица 1.6 - Операции языка

|  |  |
| --- | --- |
| Тип | Операторы |
| Арифметические | () (приоритетность операций)  + (сложение)  - (вычитание)  \* (умножение)  / (деление) |
| Сравнения | > (больше)  < (меньше)  != (не равно) |
| Сдвига | [ (сдвиг влево)  ] (сдвиг вправо) |

Наивысший приоритет среди арифметических операций присваивается умножению и делению, в то время как сложение и вычитание имеют более низкий приоритет. При равном приоритете операция, расположенная слева, выполняется в первую очередь. Порядок приоритета можно изменить, используя круглые скобки.

**1.13 Выражения и их вычисления**

Использование выражений представляет собой ключевой механизм для описания алгоритмов и является одним из фундаментальных аспектов языков программирования. Для вычисления выражений применяются определенные нормы и правила. Правила вычисления выражений языка программирования CYR-2023:

1) Выражения читаются слева направо и записываются в одну строку.

2) Реализация выражений происходит с помощью обратной польской записи.

3) Допускается деление на 0.

**1.14 Конструкции языка**

Конструкции языка CYR-2023 представлены в таблице 1.7.

Таблица 1.7 - Конструкции языка

|  |  |
| --- | --- |
| Конструкция | Представление в языке |
| Главная функция | main  {<инструкции языка>} |
| Функция | <declare> <тип данных> function <идентификатор>(<тип данных> <идентификатор>, …)  {  return<идентификатор>|<литерал>;  } |
| Блок | {  …  } |

Исходный текст программы на языке программирования CYR-2023 оформляется в виде главной функции, точки входа в программу. При составлении функций рекомендуется выделять блоки и фрагменты отступами для лучшей читаемости кода.

**1.15 Область видимости идентификаторов**

Все идентификаторы обязаны быть объявленными внутри функции. Вне функции объявление идентификаторов недопустимо. Глобальных переменных нет, только локальные. Параметры видны только внутри функции, в которой объявлены.

**1.16 Семантические проверки**

Перечень примеров семантических проверок, предусмотренных языком CYR-2023, приведен в таблице 1.8.

Таблица 1.8 - Перечень семантических проверок

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Наличие функции main |
| 2 | Имя идентификатора не может быть ключевым словом |
| 3 | Переопределение идентификаторов |
| 4 | Использование идентификаторов без их объявления |
| 5 | Проверка на количество параметров объявляемых функций и ограничения применяемые к ним (call/date) |
| 6 | Правильность передаваемых в функцию параметров: количество, типы |
| 7 | Нарушение структуры программного блока |
| 8 | Тип данных переменной должен совпадать с типом значения, которое присваивается этому типу |

Семантические проверки в языке программирования выполняются для обеспечения корректного смысла и значения кода, выявления логических ошибок, а также поддержания согласованности между различными частями программы.

**1.17 Распределение оперативной памяти на этапе выполнения**

Транслированный код использует две области памяти. В сегмент констант заносятся все литералы. В сегмент данных заносятся переменные и параметры функций. Локальная область видимости в исходном коде определяется за счет использования правил именования идентификаторов и регулируется их префиксами, что и обуславливает их локальность на уровне исходного кода, не смотря на то, что в оттранслированным в язык ассемблера коде переменные имеют глобальную область видимости.

**1.18 Стандартная библиотека и ее состав**

Содержимое библиотеки и описание функций представлено в таблице 1.9.

Таблица 1.9 - Стандартная библиотека языка CYR-2023

|  |  |
| --- | --- |
| Функция | Описание |
| void function date(); | Функция - процедура.  Принимаемых параметров нет.  Возвращаемого значения нет.  Выводит в консоль текущую дату. |
| void function time(); | Функция - процедура.  Принимаемых параметров нет.  Возвращаемого значения нет.  Выводит в консоль текущее время. |

В языке CYR-2023 присутствует стандартная библиотека, которая автоматически подключается при трансляции исходного кода в язык ассемблера. Можно пользоваться двумя видами функций стандартной библиотеки: функция вычисления времени и функция вычисления даты.

**1.19 Ввод и вывод данных**

Ввод данных не предусмотрен. Вывод данных осуществляется с помощью ключевого слова write. В качестве аргумента принимаются числовые и строковые идентификаторы, так же литералы:

write <идентификатор>|<литерал>;

**1.20 Точка входа**

В языке CYR-2023 точкой входа является ключевое слово “main”. Точка входа не может отсутствовать и должна быть единственной.

**1.21 Препроцессор**

Препроцессор для языка программирования CYR-2023 отсутствует.

**1.22 Соглашение о вызовах**

В языке используется соглашение о вызовах под названием stdcall – это значит, что в программе все параметры функции будут передаваться через стек справа налево и память будет освобождаться вызываем кодом.

**1.23 Объектный код**

Программа, исходный код которой написан на языке программирования CYR-2023, будет транслироваться в ассемблерный код, а после из ассемблерного кода в объектный код**.**

**1.24 Классификация сообщений транслятора**

В соответствии с префиксами будут различаться сообщения, представленные в таблице 1.10.

Таблица 1.10 - Классификация сообщений транслятора

|  |  |
| --- | --- |
| Интервал | Описание ошибки |
| 0 - 99 | Системные ошибки и ошибки с main/return |
| 100 - 109 | Ошибки параметров |
| 110 - 119 | Ошибки открытия и чтения файлов |
| 120 - 599 | Лексические и семантические ошибки |
| 600 - … | Синтаксические ошибки |

Транслятор генерирует сообщения об ошибках пользователю и выводит их в файл протокола.

**1.25 Контрольный пример**

Контрольный пример демонстрирует все возможности языка. Исходный код приведён в приложении А.

**2 Структура транслятора**

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

В языке программирования CYR-2023 исходный код транслируется в ассемблерный код. Транслятор языка разделен на взаимосвязанные части, которые выполняют свои определенные функции. Графическое отображение структуры транслятора представлено на рисунке 2.1.

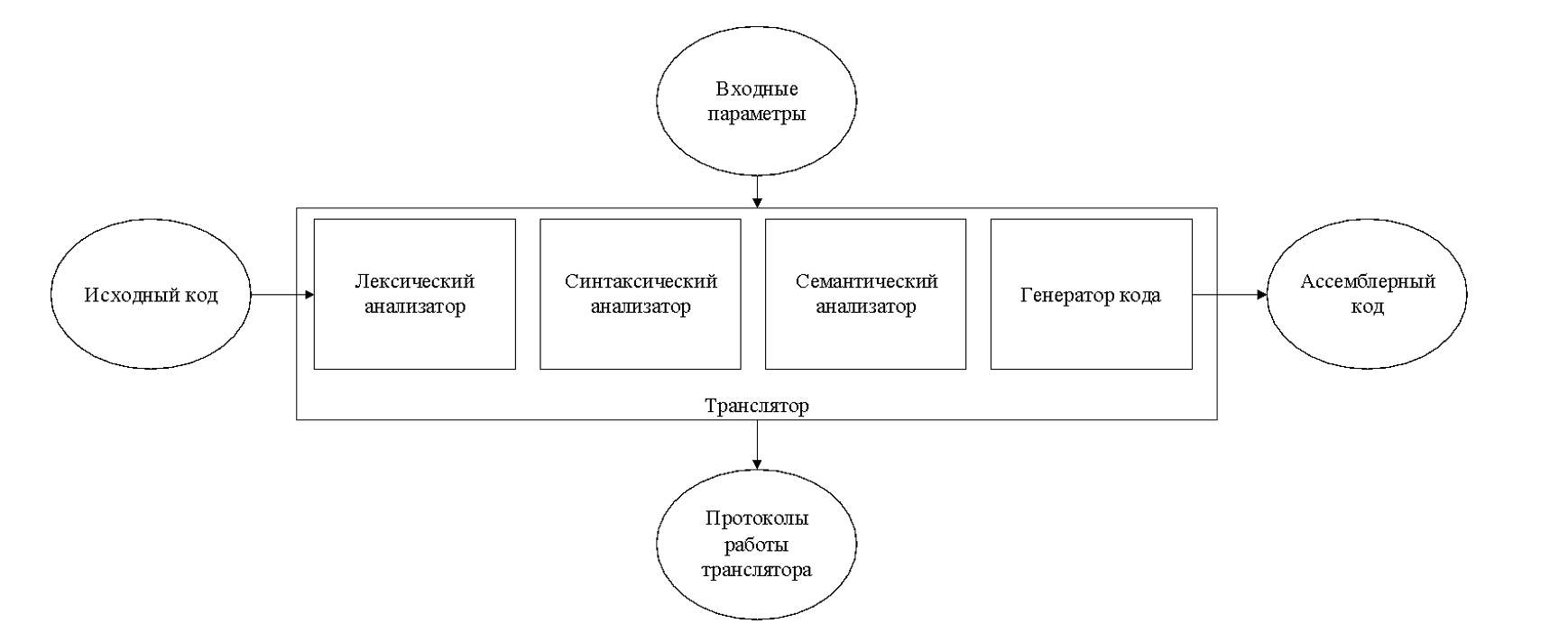


Рисунок 2.1 – Структура транслятора языка программирования CYR-2023

На первом этапе обработки исходного кода, его подвергают анализу лексическим анализатором. Этот инструмент проверяет соблюдение правил написания кода в соответствии с определенным набором допустимых цепочек языка. В результате выполнения анализа генерируются две таблицы: таблица лексем, в которой все слова исходного языка преобразуются в токены, и таблица идентификаторов. Каждой лексеме присваивается соответствующая запись в таблице идентификаторов. При этом таблица идентификаторов содержит дополнительную информацию о некоторых лексемах.

Далее наступает черед синтаксического анализатора, к нему на вход поступает таблица лексем, полученная на этапе лексического анализа. Если программа построена синтаксически правильно, то осуществляется переход к этапу семантического анализа, стоящему далее, при ином раскладе работа транслятора останавливается. Семантический анализ предполагает проверку исходного кода на семантическую согласованность конструкций языка, проверяет правильность текста исходной программы с точки зрения семантики, после чего, если код верный с точки зрения семантики, то следующий этап – это генерация кода [1].

Генерация кода реализуется посредством чистой интерпретации, без создания промежуточного представления кода. В финале происходит генерация кода, во время исполнения которого формируется объектный код.

**2.2 Перечень параметров транслятора**

Входные параметры приведены в таблице 2.1.

Таблица 2.1 – Входные параметры транслятора языка CYR-2023

|  |  |
| --- | --- |
| Параметр | Назначение |
| -in: | Указывает на файл с исходным кодом. Исходный код содержится в файле с расширением \*.txt |
| -log: | Файл для записи результата работы программы. |

Входные параметры используются для формирования файлов с информацией о результате работы транслятора.

**2.3 Протоколы, формируемые транслятором**

В ходе работы транслятора, на каждом из его этапов формируется протокол работы.

В таблице 2.2 приведен перечень протоколов, формируемых транслятором и их содержимое.

Таблица 2.2 – Протоколы, формируемые транслятором языка CYR-2023

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |
| Файл журнала, заданный параметром "-log:" | Файл содержит информация о входных параметрах приложения, этапе проверки символов на допустимость, результат работы лексического и семантического анализаторов.  При возникновении ошибок на одном из этапов, содержимое ошибки будет выведено в файл с протоколом. |
| Выходной файл, c расширением ".asm" | Результат работы программы – файл, содержащий исходный код на языке ассемблера. |

Протокол помогает контролировать прогресс, обнаруживать проблемы и обеспечивать надежность и качество процесса трансляции.

**3. Разработка лексического анализатора**

**3.1 Структура лексического анализатора**

Первый этап преобразования исходного кода называется лексическим анализом, а инструмент, осуществляющий этот анализ, называется лексическим анализатором. Лексический анализатор сканирует поток символов, составляющих исходную программу, и группирует эти символы в значимые последовательности, которые называются лексемами. Лексема – это структурная единица языка, которая состоит из элементарных символов языка и не содержит в своём составе других структурных единиц языка. Лексемами языков программирования являются идентификаторы, константы, ключевые слова языка, знаки операций и т.п.

Структура лексического анализатора представлена на рисунке 3.1.

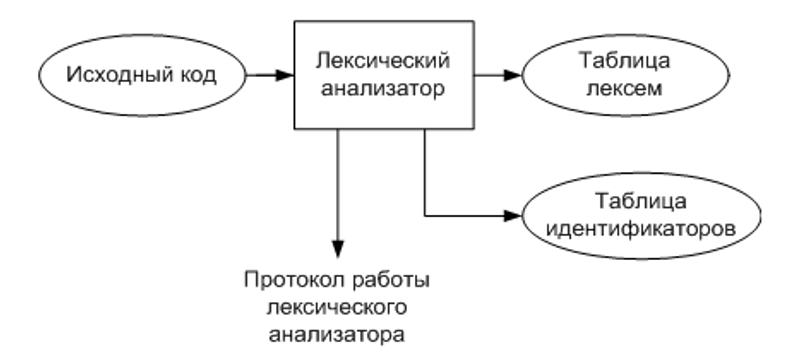


Рисунок 3.1 – Структура лексического анализатора

На вход лексического анализатора поступает текст исходной программы, а выходная информация передаётся для дальнейшей обработки синтаксическому анализатору.

**3.2. Входные и выходные данные лексического анализатора**

Исходный код на языке CYR-2023 является входными данными.

Он представляет собой текст программы, написанный на языке программирования. Лексический анализатор сканирует последовательность символов исходного кода программы, применяя заданные правила лексического анализа.

Выходные данные лексического анализатора представлены в таблице 3.1

Таблица 3.1 – Выходные данные лексического анализатора

|  |  |  |
| --- | --- | --- |
| Таблица | Определение | Содержание |
| Таблица лексем | Представляет собой структуру данных, в которой хранится информация о лексемах, выделенных лексическим анализатором. | Типы лексем, их значения, а также информация о расположении в исходном коде |
| Таблица Идентификаторов | Содержит информацию об идентификаторах, то есть символах, которые представляют переменные, функции, классы и другие сущности в программе. | Хранятся типы данных, области видимости и другие атрибуты, связанные с каждым идентификатором. |

Каждая найденная лексема сохраняется, а её тип и атрибуты могут быть определены на основе правил и таблицы символов. Это позволяет анализатору преобразовывать исходный код в последовательность лексем.

**3.3. Параметры лексического анализатора**

Параметры, управляющие работой лексического анализатора, представлены в таблице 3.2.

|  |  |  |
| --- | --- | --- |
| Параметр | Назначение | Принцип работы |
| Ключевые слова | Задает список зарезервированных слов, нельзя использовать как идентификаторы. | Лексический анализатор использует этот список для идентификации ключевых слов в коде. |
| Таблица допустимости входных символов | Определяет, какие символы считаются допустимыми в исходном коде. | Лексический анализатор различает символы, определенные в наборе, и игнорирует остальные. |
| Таблица символов | Используется для хранения информации о ранее обработанных лексемах. | Лексический анализатор использует таблицу символов для сохранения и поиска информации о символах в коде. |

Таблица 3.2 – Параметры лексического анализатора

Более подробно параметры лексического анализатора будут разобраны в разделах ниже.

**3.4. Алгоритм лексического анализатора**

Последовательность выполнения алгоритма работы лексического анализатора:

1. Считывание текста из исходного файла в буфер;
2. Далее – посимвольное считывание из буфера. Если текущий символ - конец строки – пункт 8);
3. Проверка символа на допустимость;
4. Если текущий символ сепаратор – запись в таблицу лексем. Если нет – запись в строку-буфер до тех пор, пока сепаратор не встретится;
5. Проверка строки-буфера на одно из зарезервированных слов. Если такое есть – запись в таблицу лексем и переход к пункту 2);
6. Проверка строки-буфера на соответствие одному из конечных автоматов. Если нет – пункт 8), выход из программы с ошибкой;
7. Запись в таблицу лексем соответствующего значения. Запись в таблицу идентификаторов того или иного идентификатора или литерала. Переход к пункту 2);
8. Конец работы анализатора.

**3.5. Контроль входных символов**

Исходный код на языке CYR-2023 является входными данными. Для правильного разбиения исходного текста на слова, все символы, опираясь на код символа в кодировке Windows-1251, разделяются по категориям. Таблица допустимости представлена на рисунке 3.2.

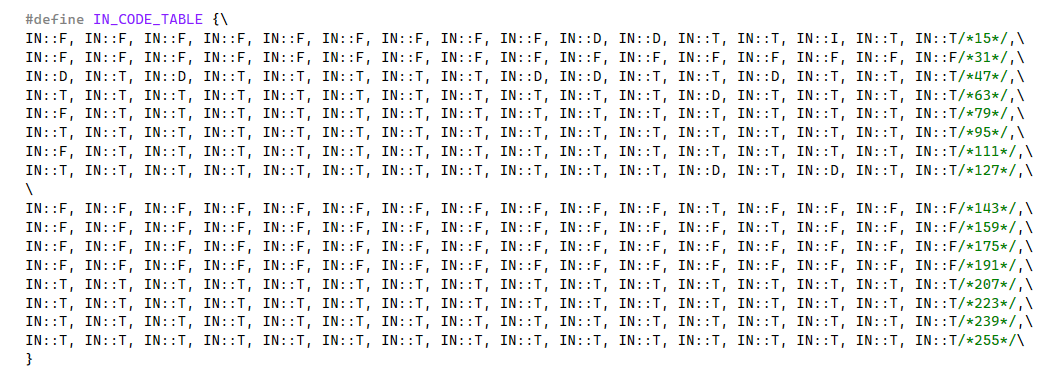


Рисунок 3.2. Таблица допустимости входных символов

Таблица допустимости была сформирована на основе кодировки Windows-1251. В таблице 3.3 представлено соответствие символов и их значений в таблице.

Таблица 3.3 – Соответствие символов и их значений в таблице

|  |  |
| --- | --- |
| Значение в таблице входных символов | Символы |
| Разрешенные алфавитом символов | T |
| Запрещенные алфавитом символов | F |
| Игнорируемые | I |
| Символы-сепараторы | D |

Таблица необходима для проверки входных символов на допустимость.

**3.6. Удаление избыточных символов**

Избыточными символами представлены пробелы и символы табуляции.

Описание алгоритма удаления избыточных символов:

1. посимвольно считываем файл с исходным кодом программы;
2. если находим лишние пробелы или знак табуляции, то переходим к следующему символу, а если находим символ перехода на новую строку, то добавляем специальный символ-сепаратор “|”;
3. продолжаем считывание файла по данному алгоритму.

**3.7. Перечень ключевых слов**

Соответствие ключевых слов и лексем приведено в таблице 3.4.

Таблица 3.4 – Перечень ключевых слов

|  |  |
| --- | --- |
| Цепочка | Лексема |
| integer | t |
| string | t |
| unsigned integer | t |
| char | t |
| function | f |
| return | r |
| main | m |
| declare | d |
| call | c |
| write | o |

Окончание таблицы 3.4 – Перечень ключевых слов

|  |  |
| --- | --- |
| Цепочка | Лексема |
| + | v |
| - | v |
| \* | v |
| / | v |
| = | = |
| ( | ( |
| ) | ) |
| { | { |
| } | } |
| , | , |
| ; | ; |
| > | > |
| < | < |
| != | ! |
| [ | [ |
| ] | ] |
| идентификатор | i |
| строковый литерал | l |

Перечень ключевых слов в лексическом анализаторе необходим для выявления ключевых конструкций языка, предотвращения возможных конфликтов идентификаторов с зарезервированными терминами, повышения читаемости и понимания кода, обеспечения поддержки синтаксического анализа и построения синтаксического дерева, а также определения областей видимости в программе.

**3.8. Основные структуры данных**

В приложении Б представлены основные структуры данных на этапе синтаксического анализа.

В таблице лексем содержатся лексемы, а также номер строки, в которой она находится. Таблица идентификаторов содержит сам идентификатор, тип данных, тип идентификатора, индекс в таблице лексем и значение.

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

Перечень примеров сообщений, генерируемых на этапе лексического анализа, представлен в таблице 3.5.

Таблица 3.5 - Сообщения лексического анализатора

|  |  |
| --- | --- |
| Код | Сообщение |
| 2 | У вас нет main |
| 3 | У вас несколько main |
| 123 | Нераспознанная лексема |
| 125 | Перезапись идентификатора |
| 126 | Неизвестная переменная |

Ошибки, возникающие на этапе лексического анализатора, фиксируются в протокол, заданный входным параметров –log. В случае возникновения ошибки производится ее запись в протокол, с точным описанием ошибки и позиции, где она возникла.

**3.10. Принцип обработки ошибок**

В случае обнаружения ошибки, транслятор прекращает свою работу и в log-файл записывается соответствующее сообщение об ошибке с указанием строки и позиции.

**3.11 Контрольный пример**

На вход лексического анализатора подается программа на языке CYR-2023, описанная в пункте 1.25. Результат работы лексического анализатора представлен в приложении В.

**4. Разработка синтаксического анализатора**

**4.1. Структура синтаксического анализатора**

Структура синтаксического анализатора языка представлена ниже на рисунке 4.1.

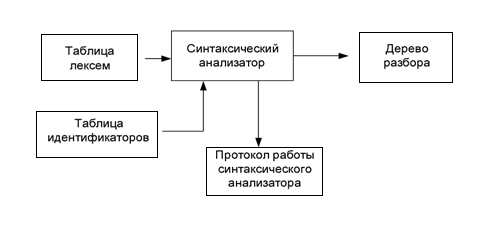


Рисунок 4.1 – Структура синтаксического анализатора.

Синтаксический анализатор представляет собой компонент компилятора, осуществляющий проверку исходного кода на соответствие правилам грамматики. Входными данными для синтаксического анализа служат таблица лексем и таблица идентификаторов. В результате работы синтаксического анализатора формируется дерево разбора [2].

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

Грамматика, описывающая язык CYR-2023 представлена в таблице 4.1.

Таблица 4.1 - грамматика языка CYR-2023

|  |  |
| --- | --- |
| Нетереминалы | Описание |
|  | Правила, описывающие общую структуру программы |
|  | Порождает правила, описывающие инструкции языка |
|  | Порождает правила, описывающие выражения |
|  | Порождает правила, описывающие арифметические действия |
|  | Порождает правила, описывающие формальные параметры функции |

Окончание таблицы 4.1 - грамматика языка CYR-2023

|  |  |
| --- | --- |
| Нетереминалы | Описание |
|  | Порождает правила, описывающие фактические параметры функции |
|  | Порождает правила, описывающие условный оператор |

Грамматика языка программирования в синтаксическом анализаторе применяется для установления правил, по которым происходит распознавание структур программы. Она формирует основу для анализа и понимания синтаксической структуры исходного кода, гарантируя правильное построение абстрактного синтаксического дерева. Грамматика также обнаруживает синтаксические ошибки и предоставляет основу для последующего анализа и выполнения программы.

**4.3. Построение конечного магазинного автомата**

Принцип действия конечного магазинного автомата представлен на рисунке 4.2.

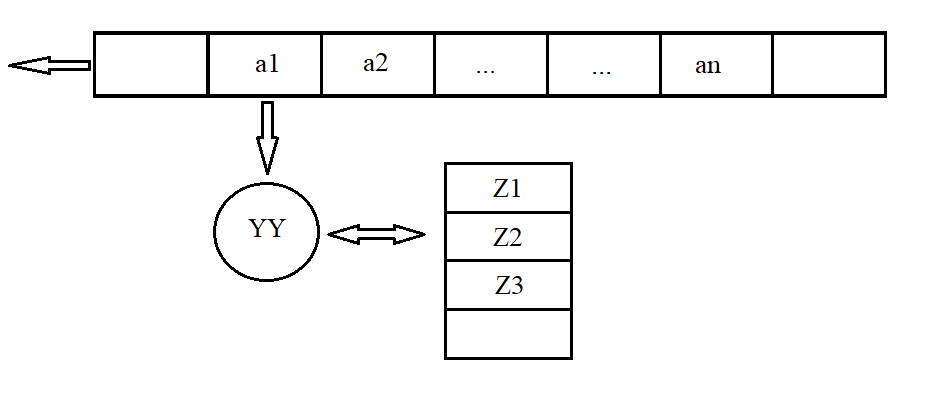


Рисунок 4.2 - МП-автомат

Формальное описание МП-автомата:



 - 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.

Цепочка  является допустимой (распознается) автоматом , если  и .

Работа автомата :

1. состояние автомата 
2. читает символ находящийся под головкой (сдвигает ленту);
3. не читает ничего (читает , не сдвигает ленту);
4. из  определяет новое состояние , если  или .
5. читает верхний (в стеке) символ  и записывает цепочку  т.к. , при этом, если , то верхний символ магазина просто удаляется.
6. работа автомата заканчивается 

**4.4. Основные структуры данных**

В приложении Г представлены основные структуры данных и правила перехода, используемые на этапе синтаксического анализа.

**4.5. Описание алгоритма синтаксического разбора**

Принцип работы автомата следующий:

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

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

Входной информацией для синтаксического анализатора является таблица лексем и идентификаторов. Кроме того, используется описание грамматики в форме Грейбах. Результаты работы синтаксического разбора, а именно дерево разбора и протокол работы автомата с магазинной памятью выводятся в журнал работы программы.

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

Перечень сообщений, генерируемых на этапе синтаксического анализа, представлен в таблице 4.3.

Таблица 4.3 - Перечень сообщений

|  |  |
| --- | --- |
| Код | Сообщение |
| 600 | Неверная структура программы |
| 601 | Ошибка в конструкции функции |
| 602 | Ошибка в выражении |
| 603 | Ошибка в параметрах функции |
| 604 | Ошибка в параметрах вызываемой функции |
| 605 | Ошибочное оператора |
| 606 | Ошибочное в условном операторе |

Перечень сообщений, генерируемых на этапе синтаксического анализа, необходим для обнаружения и документирования синтаксических ошибок в исходном коде программы.

**4.8. Принцип обработки ошибок**

Обработка ошибок осуществляется следующим образом: синтаксический анализатор проходит по всем правилам и последовательностям правил грамматики, чтобы найти соответствие с конструкцией, представленной в таблице лексем; если невозможно подобрать соответствующую последовательность, то генерируется соответствующая ошибка, которая регистрируется в протоколе работы, и выполнение программы прекращается.

**4.9. Контрольный пример**

Результатом работы синтаксического анализатора является дерево разбора. Результат работы синтаксического анализатора представлен в приложении Д.

**5. Разработка семантического анализатора**

**5.1. Структура семантического анализатора**

Структура семантического анализатора представлена на рисунке 5.1.

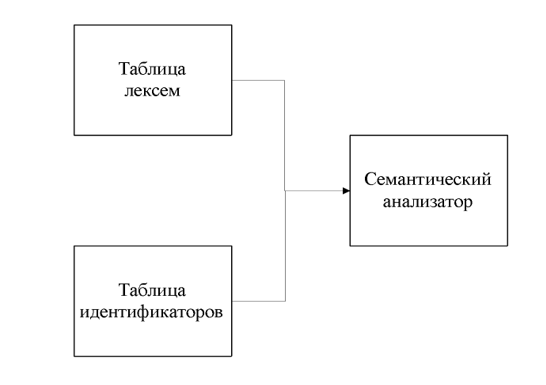


Рисунок 5.1 – Структура семантического анализатора

Семантический анализатор включает в себя ряд функций, предназначенных для проверки корректности исходной программы. Эти функции анализатора выполняются на различных этапах работы транслятора.

**5.2. Функции семантического анализатора**

Семантический анализатор выполняет проверку на основные правила языка (семантики языка), которые описаны в разделе 1.16.

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

Пример сообщений, генерируемых при выполнении семантических проверок, представлен в таблице 5.1.

Таблица 5.1 - Перечень сообщений

|  |  |
| --- | --- |
| Код | Сообщение |
| 132 | Имя идентификатора не может быть ключевым словом |
| 134 | Не найдено ключевого слова call при вызове функций даты/времени |
| 136 | Количество парамеров превышает максимальное допустимое значение |
| 137 | Неверное количество параметров date/time |
| 138 | Неверное количество параметров вызываемой функции или неверные параметры |

Окончание таблицы 5.1 - Перечень сообщений

|  |  |
| --- | --- |
| 139 | Тип функции и тип возвращаемого значения отличаются |
| 140 | Несоответствие присваиваемого типа типу переменной |
| 141 | Функция не возвращает значение |
| 142 | Нарушена структура программного блока |

Сообщения, генерируемые при выполнении семантических проверок, необходимы для выявления и уведомления о логических ошибках и непоследовательностях в программе.

**5.4. Принцип обработки ошибок**

Ошибки, возникающие в процессе трансляции программы, фиксируются в протокол, заданный входным параметром. В случае возникновения ошибок происходит их протоколирование. Все семантические ошибки являются критическими, из-за чего транслятор прекращает свою работу.

**5.5. Контрольный пример**

Результат работы семантических функций описан в главе 8.

**6. Преобразование выражений**

**6.1. Выражения, допускаемые языком**

Приоритет операций представлен в таблице 6.1. Чем выше число, тем выше и приоритет.

Таблица 6.1 – Приоритеты операций

|  |  |  |
| --- | --- | --- |
| Операция | Значение | Приоритет |
| ( ) | повышают приоритет заключенного в них выражения | 0 |
| \* | умножение | 3 |
| / | деление | 3 |
| ] | сдвиг вправо | 3 |
| [ | сдвиг влево | 3 |
| + | сложение | 2 |
| - | вычитание | 2 |

В языке CYR-2023 допускаются выражения с использованием числовых идентификаторов и литералов. Также предусмотрены следующие арифметические операции и операции сравнения: +, -, \*, /, ], [. Есть возможность изменять приоритет выполнения арифметических операций при помощи скобок (‘(…)’).

**6.2. Польская запись и принцип ее построения**

Польская запись представляет собой альтернативный способ записи арифметических выражений, отличающийся от традиционного тем, что операторы располагаются перед или после аргументов, а не между ними. Этот метод включает в себя два варианта: префиксную и постфиксную запись. Одним из основных преимуществ использования польской записи является отсутствие необходимости в использовании скобок [3].

Алгоритм построения польской записи:

1. посимвольно перебираем таблицу лексем;
2. если текущий символ – оператор или идентификатор, записываем его в финальную строку лексем и переходим к предыдущему пункту;
3. если идентификатор – функция, перед ним в финальную строку записывается токен “@”(означает вызов функции);
4. открывающаяся скобка автоматически заносится в стек операций;
5. закрывающаяся скобка выталкивает все операции из стека в финальную строку и самоуничтожается вместе с открывающейся скобкой;
6. операция выталкивает все операции с большим или равным приоритетом в финальную строку;
7. когда встречается символ конца строки кода (“;”) – все операции выталкиваются из стека в финальную строку;
8. в конце операции преобразования к польской нотации после каждой лексемы функции добавляется лексема - число параметров, передаваемых в функцию;

**6.3. Программная реализация обработки выражений**

После этапов лексического и синтаксического анализа происходит преобразование в польскую запись во время этапа генерации кода на язык ассемблера.

**6.4. Контрольный пример**

Пример разбора выражения представлен в таблице 6.2.

Таблица 6.2 - Разбор выражения

|  |  |  |
| --- | --- | --- |
| Выражение | Стек | Результат |
| (x - y)\*8 |  |  |
| x - y)\*8 | ( |  |
| - y)\*8 | ( | x |
| y)\*8 | ( - | x |
| )\*8 | ( - | x y |
| \*8 |  | x y- |
| 8 | \* | x y - |
|  | \* | x y - 8 |
|  |  | x y - 8 \* |

Преобразование выражений в формат польской записи необходимо для построения более простых алгоритмов их вычисления и преобразования к ассемблерному коду.

**7. Генерация кода**

**7.1 Структура генератора кода**

Структура генератора кода СYR-2023 представлена на рисунке 7.1.

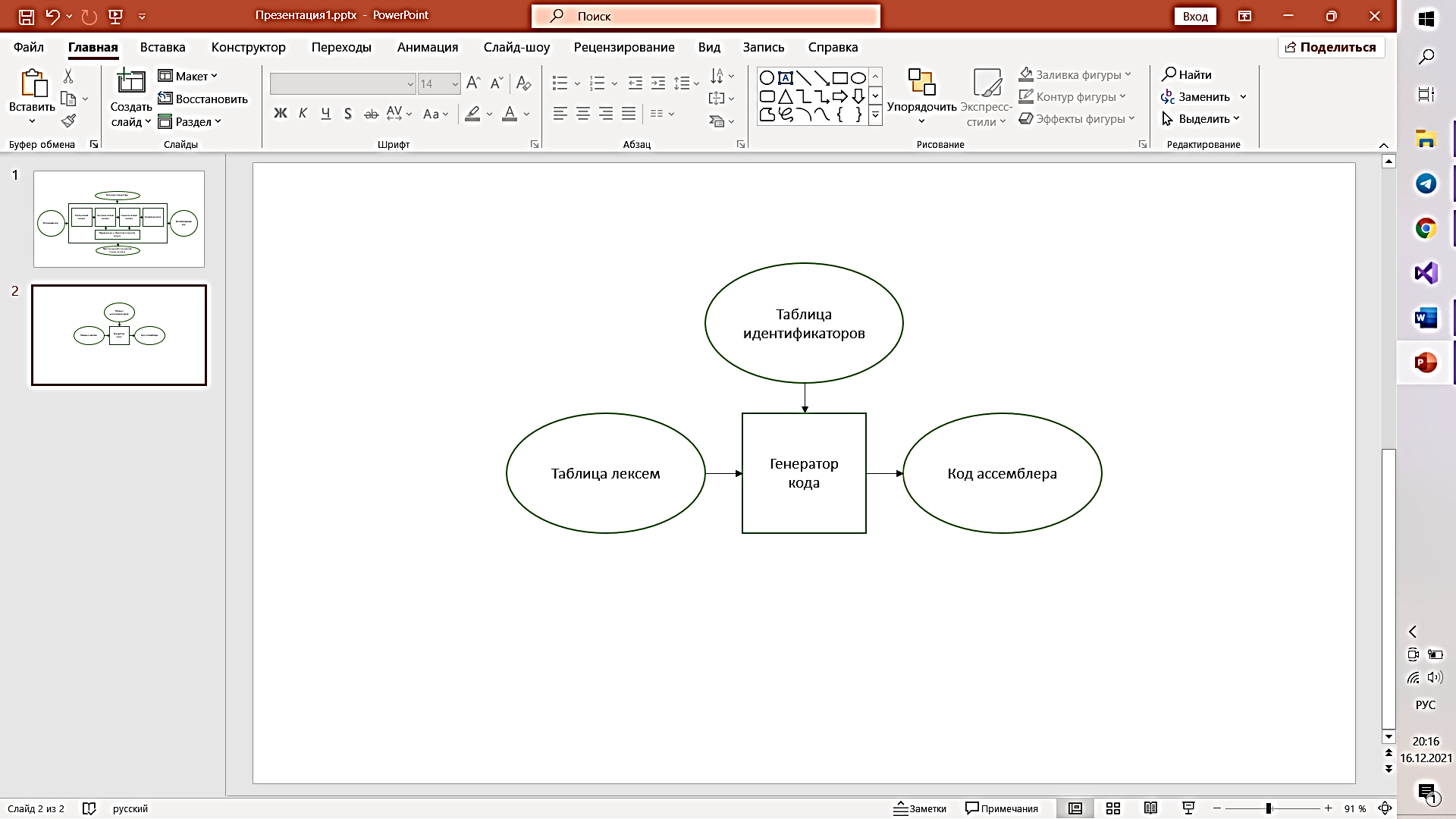


Рисунок 7.1 – Структура генератора кода

В языке CYR-2023 генерация кода является заключительным этапом трансляции. Генератор принимает на вход таблицы лексем и идентификаторов, полученные в результате лексического анализа [4]. В соответствии с таблицей лексем строится выходной файл на языке ассемблера, который будет являться результатом работы транслятора. В случае возникновения ошибок генерация кода не будет осуществляться.

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

Соответствия между типами данных идентификаторов на языке CYR-2023 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка CYR-2023 и языка ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке CYR-2023 | Тип идентификатора на языке ассемблера | Пояснение |
| int | SDWORD | Хранит целочисленный тип данных со знаком. |
| uint | DWORD | Хранит целочисленный тип данных без знака. |

Окончание таблицы 7.1 – Соответствия типов идентификаторов языка CYR-2023 и языка ассемблера

|  |  |  |
| --- | --- | --- |
| string | DWORD – для переменных | Хранит указатель на начало строки. Строка завешается нулевым символом. |
| char | BYTE – для переменных | Хранит последовательность байтов.  Переменные хранят указатель на начало строки литерала. |

Элементы таблицы идентификаторов расположены в сегментах .data и .const языка ассемблера.

**7.3. Статическая библиотека**

Статическая библиотека написана на языке программирования C++. Библиотека подключается с помощью команды includelib <имя.lib>. В процессе компоновки статическая библиотека подключается к выходному файлу.

Подробное описание функций приведено в пункте 1.18.

**7.4. Особенности алгоритма генерации кода**

Общая схема работы генератора кода представлена на рисунке 7.2.

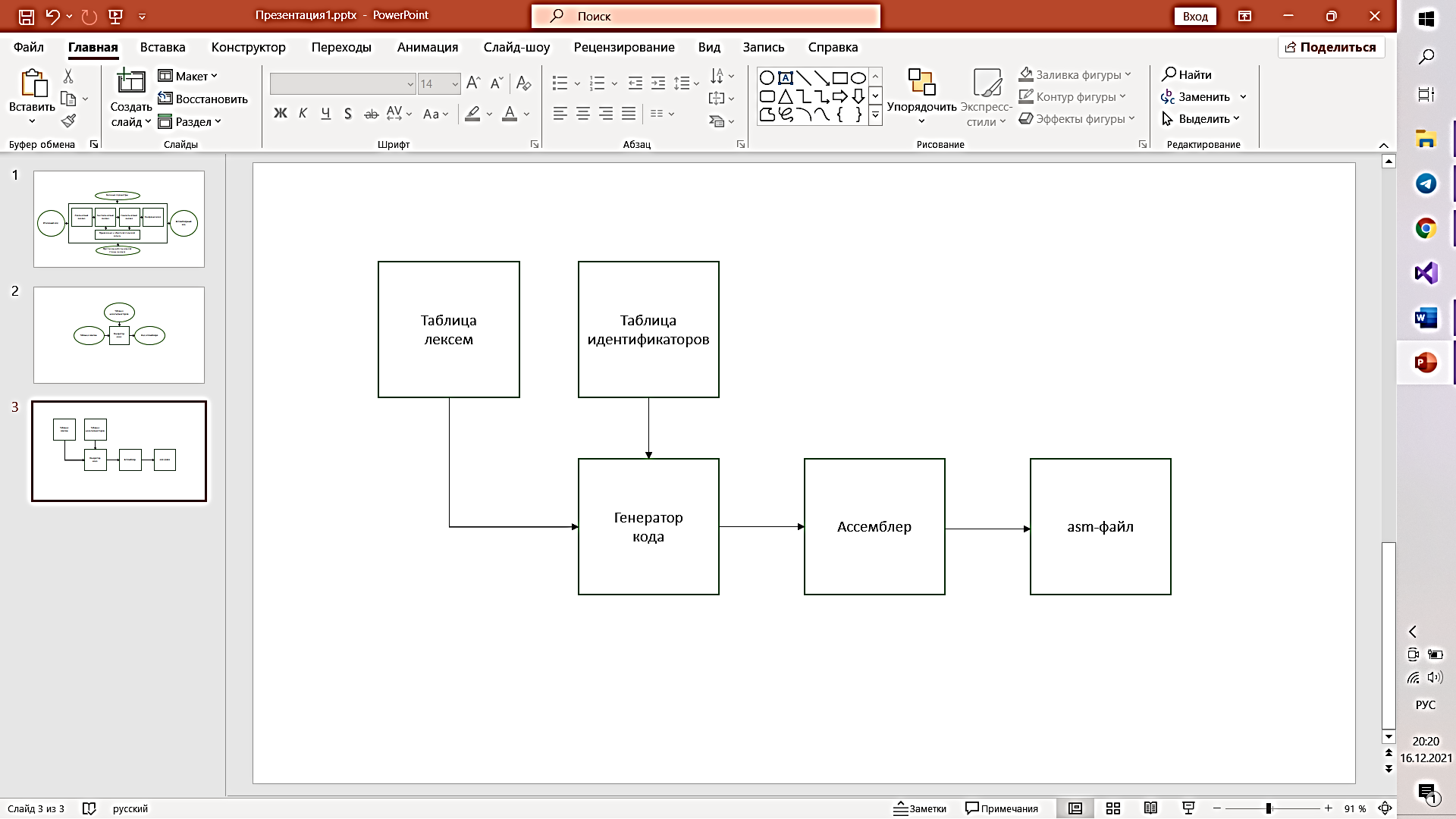


Рисунок 7.2 – Структура генератора кода

В языке CYR-2023 генерация кода происходит на основе таблиц лексем и таблиц идентификаторов.

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

На вход генератору кода поступают таблицы лексем и идентификаторов исходного кода программы на языке CYR-2023. Результаты работы генератора кода выводятся в файл с расширением .asm [5].

**7.6. Контрольный пример**

Генерируемый код записывается в файл ассемблера с расширением .asm. Сгенерированный код можно посмотреть в приложении Е.

**8. Тестирование транслятора**

**8.1. Общие положения**

При возникновении ошибки она обрабатывается в основном файле программы: детали ошибки записываются в файл журнала сборки. В случае отсутствия ошибок в журнале отобразится полная информация о проекте, и проект успешно скомпилируется. На этапе трансляции могут возникнуть ошибки разного характера. Ниже приведен пример списка ошибок, которые могут возникнуть на различных этапах трансляции, на примере кода с заранее допущенными ошибками.

**8.2 Результаты тестирования**

Результат тестирования лексического анализатора приведен в таблице 8.1.

Таблица 8.1 – Тестирование лексического анализатора

|  |  |
| --- | --- |
| Код с лексической ошибкой | Сообщение о допущенной лексической ошибке |
| main  {  declare integer e;  e = 7;  declare integer e;  return e;  }; | Ошибка 125: Перезапись идентификатора |

Тестирование ошибок на этапе синтаксического анализатора представлено в таблице 8.2.

Таблица 8.2 – Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Код с синтаксической ошибкой | Сообщение о допущенной синтаксической ошибке |
| main  {  declare integer x = 1;  declare integer z;  return 0;  };  main  {  declare integer p = 1;  return 0;  }; | Ошибка 3: У вас несколько main |

Тестирование ошибок на этапе семантического анализатора представлено в таблице 8.3.

Таблица 8.3 – Тестирование семантического анализатора

|  |  |
| --- | --- |
| Код с семантической ошибкой | Сообщение о допущенной семантической ошибке |
| main  {  declare integer x;  x = 'hello!';  return 0;  }; | Ошибка 140: Несоответствие присваемого типа типу переменной |

Тестирование в программировании нужно для обнаружения и исправления ошибок, подтверждения правильной работы.

**Заключение**

В ходе выполнения курсовой работы был разработан полноценный транслятор для языка программирования CYR-2023.

Таким образом, были выполнены основные задачи курсовой работы:

1. разработка спецификации языка программирования CYR-2023;
2. разработка структуры транслятора;
3. разработка лексического распознавателя;
4. разработка синтаксического распознавателя;
5. разработка семантического распознавателя;
6. разработка преобразования выражений в форму обратной польской записи;
7. разработка генератора кода в Assembler;
8. выполнено тестирования работоспособности транслятора.

Окончательная версия языка CYR-2023 включает:

1. 4 типа данных;
2. Наличие 4 арифметических операторов для вычисления выражений;
3. Реализация операций сдвига;
4. Реализация операции вывода write;
5. Реализована система обработки ошибок;
6. Реализация функций стандартной библиотеки (date/time);
7. Реализация условного оператора;
8. Возможность инициализации при объявлении переменной.

Знания, приобретенные в ходе выполнения курсового проекта, будут способствовать более эффективному освоению новых технологий. Это связано с тем, что изучение новых языков программирования будет осуществляться уже на уровне глубокого понимания их функциональности, а не ограничится простым изучением синтаксиса.

**Список используемых источников**

1. Принципы работы транслятора [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/435520/. – Дата доступа:29.11.2023.

2. Ахо, А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003. – 768 с.

3. Прата, С. Язык программирования С++. Лекции и упражнения / С. Прата. – М., 2006 — 1104 c.

4. Страуструп, Б. Принципы и практика использования C++ / Б. Страуструп – 2009 – 1238 с.

5. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. - 3-е изд. – Москва : Вильямс, 2003. – 429 с.

# **Приложение А**

declare integer function one(integer a, integer b)

{

declare integer res = 1+a\*b;

return res;

};

main

{

call date();

call time();

declare integer fa = false;

declare integer fb = true;

if(fa>fb;)

{

write 'True';

};

write fa;

write fb;

write 'двоичная система счисления:';

declare integer g = 2x101;

write g;

write 'восьмеричная система счисления:';

declare unsigned integer k = 20;

write k;

write 'sdvig vlevo';

declare integer sdvl = 2x11 [ 2;

write sdvl;

write 'sdvig vpravo';

declare integer sdvr = 2x1100 ] 2;

write sdvr;

write '--';

declare string mstr = 'string!!';

write mstr;

write 'char:';

declare char aa = 'A';

write aa;

declare integer fres;

write 'function:';

fres = one(fa,fb);

write fres;

return 0;

}

Листинг 1 - Исходный код на языке CYR-2023

# **Приложение Б**

|  |
| --- |
| namespace IT // таблица идентификаторов  {  enum IDDATATYPE  {  DEF = 0, UNS = 1, INT = 2, STR = 3, VOI = 4, CHR = 5  }; // типы данных идентификаторов: integer, string  enum IDTYPE { D = 0, V = 1, F = 2, P = 3, L = 4, OP = 5, VF = 6 };  // типы идентификаторов: переменная, функция, параметр, литерал, оператор, переменная в условном операторе  struct Entry // строка таблицы идентификаторов  {  Int idxfirstLE; // индекс первой строки в таблице лексем  char visibility[ID\_MAXSIZE];// область видимости  char id[ID\_MAXSIZE];  IDDATATYPE iddatatype; // тип данных  IDTYPE idtype; // тип идентификатора  int countOfPar = 0;  union  {  int vint; // значение integer  unsigned int vuint; // значение unsigned integer  char operation; //операция  struct  {  int len; // количество символов в string  char str[TI\_STR\_MAXSIZE - 1]; // символы string  } vstr; // значение string  } value; // значение идентификатора  };  struct IdTable // экземпляр таблицы идентификаторов  {  int maxsize; // емкость таблицы идентификаторов < TI\_MAXSIZE  int size; // текущий размер таблицы идентификаторов < maxsize  Entry\* table; // массив строк таблицы идентификаторов  };  IdTable Create( // создать таблицу идентификаторов  int size // емкость таблицы идентификаторов < TI\_MAXSIZE  );  void Add( // добавить строку в таблицу идентификаторов  IdTable& idtable, // экземпляр таблицы идентификаторов  Entry entry // строка таблицы идентификаторов  );  Entry GetEntry( // получить строку таблицы идентификаторов  IdTable& idtable, // экземпляр таблицы идентификаторов  int n // номер получаемой строки  );  int IsId( // возврат: номер строки (если есть), TI\_NULLIDX (если нет)  IdTable& idtable, // экземпляр таблицы идентификаторов  char id[ID\_MAXSIZE] // идентификатор  );  void Delete(IdTable& idtable); // удалить таблицу лексем (освободить память)  void showTable(std::ostream\* mr, IdTable& idtable);  };  namespace LT  {  struct Entry //структура лексемы  {  char lexema; //лексема  int sn; //номер строки в исходном тексте  int idxTI = -1; //индекс в ТИ  int priority;  int position;  Entry(const char lex, int str\_n, int idxTI = -1)  {  this->lexema = lex;  this->sn = str\_n;  this->idxTI = idxTI;  }  Entry()  {  this->lexema = '\0';  this->sn = 0;  this->idxTI = -1;  this->priority = 0;  this->position = 0;  }  };  struct LexTable //структура таблицы лексем  {  int maxsize; //максимальный размер  int size; //текущий размер  Entry\* table; //лексема в таблице  };  LexTable Create(int size);  //функция создания таблицы лексем с параметром МАКСИМАЛЬНЫЙ РАЗМЕР  void Add(LexTable& lextable, Entry entry);  void Add(LexTable& lextable, Entry entry, int i);  Entry GetEntry(LexTable& lextable, int n);  void Delete(LexTable& lextable);  Entry writeEntry(Entry& entry, char lexema, int indx, int line);  void showTable(LexTable lextable, Parm::PARM parm);  void showTablePolish(LexTable lextable, Parm::PARM parm);  } |

Листинг 2 – Структуры данных, использованные на этапе лексического анализа

# **Приложение В**

|  |
| --- |
| 1. dtfi(ti,ti)  2. {  3. dti=lvivi;  4. ri;  5. };  7. m{  8. ci();  9. ci();  10. dti=l;  11. dti=l;  12. ?(i>i;)  13. {  14. ol;  15. };  16. oi;  17. oi;  18. ol;  19. dti=l;  20. oi;  21. ol;  22. dti=l;  23. oi;  24. ol;  25. dti=lvl;  26. oi;  27. ol;  28. dti=lvl;  29. oi;  30. ol;  31. dti=l;  32. oi;  33. ol;  34. dti=l;  35. oi;  36. dti;  37. ol;  38. i=i(i,i);  39. oi;  40. rl;  41. } |

Листинг 3 – Таблица лексем контрольного примера

---------------------------------------------------------------------------------------

№ | Идентификатор | Тип данных | Тип идентификатора | Индекс в ТЛ | Значение

---------------------------------------------------------------------------------------

0000 | one | integer | функция | 3 | -

0001 | onea | integer | параметр | 6 | -

0002 | oneb | integer | параметр | 9 | -

0003 | oneres | integer | переменная | 14 | 0

0004 | L1 | integer | литерал | 16 | 1

0005 | + | - | оператор | 17 | -

0006 | \* | - | оператор | 19 | -

0007 | main | - | функция | 27 | -

0008 | date | void | функция | 30 | -

0009 | time | void | функция | 35 | -

0010 | mainfa | integer | переменная | 41 | 0

0011 | L2 | integer | литерал | 43 | 0

0012 | mainfb | integer | переменная | 47 | 0

0013 | L3 | integer | литерал | 49 | 1

0014 | if | - | оператор | 51 | -

0015 | L4 | string | литерал | 60 | [4]"True"

0016 | L5 | string | литерал | 71 | [27]"двоичная система счисления:"

0017 | maing | integer | переменная | 75 | 0

0018 | L6 | integer | литерал | 77 | 5

0019 | L7 | string | литерал | 83 | [31]"восьмеричная система счисления:"

0020 | maink | unsigned | переменная | 87 | -

0021 | L8 | integer | литерал | 89 | 16

0022 | L9 | string | литерал | 95 | [11]"sdvig vlevo"

0023 | mainsdvl | integer | переменная | 99 | 0

0024 | L10 | integer | литерал | 101 | 3

0025 | [ | - | оператор | 102 | -

0026 | L11 | integer | литерал | 103 | 2

0027 | L12 | string | литерал | 109 | [12]"sdvig vpravo"

0028 | mainsdvr | integer | переменная | 113 | 0

0029 | L13 | integer | литерал | 115 | 12

0030 | ] | - | оператор | 116 | -

0031 | L14 | integer | литерал | 117 | 2

0032 | L15 | string | литерал | 123 | [2]"--"

0033 | mainmstr | string | переменная | 127 | [0]""

0034 | L16 | string | литерал | 129 | [8]"string!!"

0035 | L17 | string | литерал | 135 | [5]"char:"

0036 | mainaa | char | переменная | 139 | [5]""

0037 | L18 | char | литерал | 141 | [1]"A"

0038 | mainfres | integer | переменная | 148 | 0

0039 | L19 | string | литерал | 151 | [9]"function:"

0040 | L20 | integer | литерал | 166 | 7293400

---------------------------------------------------------------------------------------

Количество идентификаторов: 36

---------------------------------------------------------------------------------------

Листинг 4 – Таблица идентификаторов контрольного примера

# **Приложение Г**

|  |
| --- |
| #pragma once  #include "Greibach.h"  #define GRB\_ERROR\_SERIES 600  #define NS(n) GRB::Rule::Chain::N(n)  #define TS(n) GRB::Rule::Chain::T(n)  #define ISNS(n) GRB::Rule::Chain::isN(n)  namespace GRB  {  Greibach greibach( //создание элемента структуры с помощью конструктора с параметрами  NS('S'), TS('$'), // стартовый символ S, дно стека $  7, // количество правил  Rule(  NS('S'), GRB\_ERROR\_SERIES + 0, // первое правило - неверная структура программы(мы проверяем если все таки неверная то потом выкидываем ошибку)  4, //  Rule::Chain(8, TS('m'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';')),  Rule::Chain(15, TS('d'), TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';'), NS('S')),  Rule::Chain(14, TS('d'), TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';')),  Rule::Chain(9, TS('m'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), TS(';'), NS('S'))  ),  Rule(  NS('N'), GRB\_ERROR\_SERIES + 1, // конструкции в функциях  21, // количество правил  Rule::Chain(5, TS('d'), TS('t'), TS('i'), TS(';'), NS('N')),  Rule::Chain(4, TS('i'), TS('='), NS('E'), TS(';')),  /\* Rule::Chain(4, TS('i'), TS('('), TS(')'), TS(';')),\*/  Rule::Chain(5, TS('c'), TS('i'), TS('('), TS(')'), TS(';')),  Rule::Chain(6, TS('c'), TS('i'), TS('('), TS(')'), TS(';'), NS('N')),  Rule::Chain(6, TS('c'), TS('i'), TS('('), NS('W'), TS(')'), TS(';')),  Rule::Chain(7, TS('c'), TS('i'), TS('('), NS('W'), TS(')'), TS(';'), NS('N')),  /\*Rule::Chain(5, TS('i'), TS('('), TS(')'), TS(';'), NS('N')),\*/  Rule::Chain(5, TS('i'), TS('='), NS('E'), TS(';'), NS('N')),  Rule::Chain(9, TS('d'), TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS(';'), NS('N')),  /\* Rule::Chain(8, TS('d'), TS('t'), TS('f'), TS('i'), TS('('), TS(')'), TS(';'), NS('N')),\*/  Rule::Chain(4, TS('o'), TS('i'), TS(';'), NS('N')),  Rule::Chain(4, TS('o'), TS('l'), TS(';'), NS('N')),  Rule::Chain(4, TS('o'), NS('E'), TS(';'), NS('N')),  Rule::Chain(3, TS('o'), NS('E'), TS(';')),  Rule::Chain(4, TS('d'), TS('t'), TS('i'), TS(';')),  Rule::Chain(7, TS('d'), TS('t'), TS('i'), TS('='), NS('E'), TS(';'), NS('N')),  Rule::Chain(6, TS('d'), TS('t'), TS('i'), TS('='), NS('E'), TS(';')),  Rule::Chain(8, TS('d'), TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS(';')),  Rule::Chain(7, TS('d'), TS('t'), TS('f'), TS('i'), TS('('), TS(')'), TS(';')),  Rule::Chain(3, TS('o'), TS('i'), TS(';')),  Rule::Chain(3, TS('o'), TS('l'), TS(';')),  Rule::Chain(10, TS('?'), TS('('), NS('E'), NS('I'), TS(';'), TS(')'), TS('{'), NS('N'), TS('}'), TS(';')), //оператор if  Rule::Chain(11, TS('?'), TS('('), NS('E'), NS('I'), TS(';'), TS(')'), TS('{'), NS('N'), TS('}'), TS(';'), NS('N')) //  ),  Rule(  NS('E'), GRB\_ERROR\_SERIES + 2, // ошибка в выражении  8, //  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l')),  Rule::Chain(3, TS('('), NS('E'), TS(')')),  Rule::Chain(4, TS('i'), TS('('), NS('W'), TS(')')),  /\*Rule::Chain(3, TS('i'), TS('('), TS(')')),\*/  Rule::Chain(2, TS('i'), NS('M')),  Rule::Chain(2, TS('l'), NS('M')),  Rule::Chain(4, TS('('), NS('E'), TS(')'), NS('M')),  Rule::Chain(5, TS('i'), TS('('), NS('W'), TS(')'), NS('M'))  /\* Rule::Chain(4, TS('i'), TS('('), TS(')'), NS('M'))\*/  ),  Rule(  NS('F'), GRB\_ERROR\_SERIES + 3, // ошибка в параметрах функции  2, //  Rule::Chain(2, TS('t'), TS('i')),  Rule::Chain(4, TS('t'), TS('i'), TS(','), NS('F'))  ),  Rule(  NS('W'), GRB\_ERROR\_SERIES + 4, // ошибка в параметрах вызываемой функции  4, //  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l')),  Rule::Chain(3, TS('i'), TS(','), NS('W')),  Rule::Chain(3, TS('l'), TS(','), NS('W'))  ),  Rule(  NS('M'), GRB\_ERROR\_SERIES + 5, // оператор  2, //  Rule::Chain(2, TS('v'), NS('E')),  Rule::Chain(3, TS('v'), NS('E'), NS('M'))  ),  Rule(  NS('I'), GRB\_ERROR\_SERIES + 6, // ошибка в условном операторе  3, //  Rule::Chain(2, TS('!'), NS('E')),  Rule::Chain(2, TS('<'), NS('E')),  Rule::Chain(2, TS('>'), NS('E'))  )  );  } |

Листинг 5 – Структура данных грамматики Грейбах

# **Приложение Д**

0---: S->dtfi(F){NrE;};S--

5---: F->ti,F-------------

8---: F->ti---------------

12--: N->dti=E;-----------

16--: E->lM---------------

17--: M->vE---------------

18--: E->iM---------------

19--: M->vE---------------

20--: E->i----------------

23--: E->i----------------

27--: S->m{NrE;};---------

29--: N->ci();N-----------

34--: N->ci();N-----------

39--: N->dti=E;N----------

43--: E->l----------------

45--: N->dti=E;N----------

49--: E->l----------------

51--: N->?(EI;){N};N------

53--: E->i----------------

54--: I->>E---------------

55--: E->i----------------

59--: N->oE;--------------

60--: E->l----------------

64--: N->oi;N-------------

67--: N->oi;N-------------

70--: N->ol;N-------------

73--: N->dti=E;N----------

77--: E->l----------------

79--: N->oi;N-------------

82--: N->ol;N-------------

85--: N->dti=E;N----------

89--: E->l----------------

91--: N->oi;N-------------

94--: N->ol;N-------------

97--: N->dti=E;N----------

101-: E->lM---------------

102-: M->vE---------------

103-: E->l----------------

105-: N->oi;N-------------

108-: N->ol;N-------------

111-: N->dti=E;N----------

115-: E->lM---------------

116-: M->vE---------------

117-: E->l----------------

119-: N->oi;N-------------

122-: N->ol;N-------------

125-: N->dti=E;N----------

129-: E->l----------------

131-: N->oi;N-------------

134-: N->ol;N-------------

137-: N->dti=E;N----------

141-: E->l----------------

143-: N->oi;N-------------

146-: N->dti;N------------

150-: N->ol;N-------------

153-: N->i=E;N------------

155-: E->i(W)-------------

157-: W->i,W--------------

159-: W->i----------------

162-: N->oE;--------------

163-: E->i----------------

166-: E->l----------------

Листинг 6 – Дерево разбора

# **Приложение Е**

.586

.model flat, stdcall

includelib Libryary.lib

includelib kernel32.lib

includelib libucrt.lib

ExitProcess PROTO : DWORD

Date PROTO

Time PROTO

Abs PROTO

outputstr PROTO : DWORD

outputuint PROTO : DWORD

outputint PROTO : SDWORD

.stack 4096

.const

overflow db 'True', 0

null\_division db 'ERROR: DIVISION BY ZERO', 0

oneL1 SDWORD 1

mainL2 SDWORD 0

mainL3 SDWORD 1

mainL4 BYTE "True", 0

mainL5 BYTE "двоичная система счисления:", 0

mainL6 SDWORD 5

mainL7 BYTE "восьмеричная система счисления:", 0

mainL8 SDWORD 16

mainL9 BYTE "sdvig vlevo", 0

mainL10 SDWORD 3

mainL11 SDWORD 2

mainL12 BYTE "sdvig vpravo", 0

mainL13 SDWORD 12

mainL14 SDWORD 2

mainL15 BYTE "--", 0

mainL16 BYTE "string!!", 0

mainL17 BYTE "char:", 0

mainL18 BYTE "A", 0

mainL19 BYTE "function:", 0

mainL20 SDWORD 9390000

.data

oneoneres SDWORD 0

mainmainfa SDWORD 0

mainmainfb SDWORD 0

mainmaing SDWORD 0

mainmaink SDWORD 0

mainmainsdvl SDWORD 0

mainmainsdvr SDWORD 0

mainmainmstr DWORD ?

mainmainaa DWORD ?

mainmainfres SDWORD 0

.code

one\_proc PROC, oneoneb : SDWORD, oneonea : SDWORD

push oneL1

push oneonea

push oneoneb

pop eax

pop ebx

imul eax, ebx

jo EXIT\_OVERFLOW

push eax

pop eax

pop ebx

add eax, ebx

jo EXIT\_OVERFLOW

push eax

pop oneoneres

push oneoneres

jmp EXIT

EXIT\_DIV\_ON\_NULL:

push offset null\_division

call outputstr

push - 1

call ExitProcess

EXIT\_OVERFLOW:

push offset overflow

call outputstr

push - 2

call ExitProcess

EXIT:

pop eax

ret 8

one\_proc ENDP

main PROC

call Date

push eax

call Time

push eax

push mainL2

pop mainmainfa

push mainL3

pop mainmainfb

push mainmainfa

pop ebx

push mainmainfb

pop eax

cmp ebx, eax

ja metka112

jmp metka212

metka112:

push offset mainL4

call outputstr

metka212:

push mainmainfa

call outputint

push mainmainfb

call outputint

push offset mainL5

call outputstr

push mainL6

pop mainmaing

push mainmaing

call outputint

push offset mainL7

call outputstr

push mainL8

pop mainmaink

push mainmaink

call outputuint

push offset mainL9

call outputstr

push mainL10

push mainL11

pop ebx

pop eax

mov cl, bl

shl eax, cl

push eax

pop mainmainsdvl

push mainmainsdvl

call outputint

push offset mainL12

call outputstr

push mainL13

push mainL14

pop ebx

pop eax

mov cl, bl

shr eax, cl

push eax

pop mainmainsdvr

push mainmainsdvr

call outputint

push offset mainL15

call outputstr

push offset mainL16

pop mainmainmstr

push dword ptr mainmainmstr

call outputstr

push offset mainL17

call outputstr

mov mainmainaa,offset mainL18

push dword ptr mainmainaa

call outputstr

push offset mainL19

call outputstr

push mainmainfa

push mainmainfb

call one\_proc

push eax

pop mainmainfres

push mainmainfres

call outputint

jmp EXIT

EXIT\_DIV\_ON\_NULL:

push offset null\_division

call outputstr

push - 1

call ExitProcess

EXIT\_OVERFLOW:

push offset overflow

call outputstr

push - 2

call ExitProcess

EXIT:

push 0

call ExitProcess

main ENDP

end main

Листинг 7 – Результат генерации кода