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 KVV-2024»

Выполнил студент Качинскас Вацловас Вацловович

(Ф.И.О.)

Руководитель проекта Наркевич Аделина Сергеевна

Заведующий кафедрой к.т.н., доц. Смелов Владимир Владиславович

Консультанты Наркевич Аделина Сергеевна

Нормоконтролер Наркевич Аделина Сергеевна

Курсовой проект защищен с оценкой

Минск 2024

Содержание

Оглавление

[**Введение** 5](#_Toc185475936)

[1. Спецификация языка программирования 6](#_Toc185475937)

[1.1. Характеристика языка программирования 6](#_Toc185475938)

[1.2. Определение алфавита языка программирования 6](#_Toc185475939)

[1.3. Применяемые сепараторы 6](#_Toc185475940)

[1.4. Применяемые кодировки 6](#_Toc185475941)

[1.5. Типы данных 7](#_Toc185475942)

[1.6. Преобразование типов данных 7](#_Toc185475943)

[1.7. Идентификаторы 7](#_Toc185475944)

[1.8. Литералы 8](#_Toc185475945)

[1.9. Объявление данных 8](#_Toc185475946)

[1.10. Инициализация данных 8](#_Toc185475947)

[1.11. Инструкции языка 9](#_Toc185475948)

[1.12. Операции языка 9](#_Toc185475949)

[1.13. Выражения и их вычисление 10](#_Toc185475950)

[1.14 Конструкции языка 10](#_Toc185475951)

[1.15 Область видимости идентификаторов 11](#_Toc185475952)

[1.16 Семантические проверки 11](#_Toc185475953)

[1.17 Распределение оперативной памяти на этапе выполнения 11](#_Toc185475954)

[1.18 Стандартная библиотека и ее состав 11](#_Toc185475955)

[1.19. Ввод и вывод данных 11](#_Toc185475956)

[1.20 Точка входа 12](#_Toc185475957)

[1.21 Препроцессор 12](#_Toc185475958)

[1.22 Соглашение о вызовах 12](#_Toc185475959)

[1.23 Объектный код 12](#_Toc185475960)

[1.24 Классификация сообщений транслятора 12](#_Toc185475961)

[1.25 Контрольный пример 12](#_Toc185475962)

[2. Структура транслятора 13](#_Toc185475963)

[2.1 Компоненты транслятора, их назначение и принципы взаимодействия 13](#_Toc185475964)

[2.2 Перечень параметров транслятора 13](#_Toc185475965)

[2.3 Протоколы, формируемые транслятором 14](#_Toc185475966)

[3. Разработка лексического анализатора 15](#_Toc185475967)

[3.1 Структура лексического анализатора 15](#_Toc185475968)

[3.2 Контроль входных символов 15](#_Toc185475969)

[3.3 Удаление избыточных символов 16](#_Toc185475970)

[3.4 Перечень ключевых слов 17](#_Toc185475971)

[3.5 Основные структуры данных 17](#_Toc185475972)

[3.6 Структура и перечень сообщений лексического анализатора 19](#_Toc185475973)

[3.7 Принцип обработки ошибок 19](#_Toc185475974)

[3.8 Параметры лексического анализатора 19](#_Toc185475975)

[3.9 Алгоритм лексического анализа 20](#_Toc185475976)

[3.10 Контрольный пример 20](#_Toc185475977)

[4. Разработка синтаксического анализатора 21](#_Toc185475978)

[4.1 Структура синтаксического анализатора 21](#_Toc185475979)

[4.2 Контекстно-свободная грамматика, описывающая синтаксис языка 21](#_Toc185475980)

[4.3 Построение конечного магазинного автомата 23](#_Toc185475981)

[4.4 Основные структуры данных 24](#_Toc185475982)

[4.5 Описание алгоритма синтаксического разбора 24](#_Toc185475983)

[4.6 Структура и перечень сообщений синтаксического анализатора 25](#_Toc185475984)

[4.7 Параметры синтаксического анализатора и режимы его работы 25](#_Toc185475985)

[4.8 Принцип обработки ошибок 25](#_Toc185475986)

[4.9 Контрольный пример 26](#_Toc185475987)

[5. Разработка семантического анализатора 27](#_Toc185475988)

[5.1 Структура семантического анализатора 27](#_Toc185475989)

[5.2 Функции семантического анализатора 27](#_Toc185475990)

[5.3 Структура и перечень сообщений семантического анализатора 27](#_Toc185475991)

[5.4 Принцип обработки ошибок 28](#_Toc185475992)

[5.5 Контрольный пример 28](#_Toc185475993)

[6. Вычисление выражений 29](#_Toc185475994)

[6.1 Выражения, допускаемые языком 29](#_Toc185475995)

[7. Генерация кода 31](#_Toc185475996)

[7.1 Структура генератора кода 31](#_Toc185475997)

[7.2 Представление типов данных в оперативной памяти 31](#_Toc185475998)

[7.3 Статическая библиотека 32](#_Toc185475999)

[8. Тестирование транслятора 36](#_Toc185476000)

[**Заключение** 38](#_Toc185476001)

[**Список используемых источников** 39](#_Toc185476002)

[ПРИЛОЖЕНИЕ В 57](#_Toc185476003)

[ПРИЛОЖЕНИЕ Г 60](#_Toc185476004)

[ПРИЛОЖЕНИЕ Д 63](#_Toc185476005)

[ПРИЛОЖЕНИЕ Д 66](#_Toc185476006)

[ПРИЛОЖЕНИЕ Ж 68](#_Toc185476007)

[ПРИЛОЖЕНИЕ З 73](#_Toc185476008)

**Введение**

Целью курсового проекта является разработка собственного языка программирования KVV-2024 и компилятора для его обработки. Этот язык является учебным и будет состоять из набора простейших операций над типами данных и функций.

Для создания собственного языка программирования необходимо определить его алфавит, а также набор правил, определяющих синтаксис и семантику языка. Описание языка будет представлено в виде формы Бэкуса-Наура.

Для преобразования исходного языка в машинный необходима организация работы транслятора. В его основные задачи входят лексический анализ, синтаксический анализ, семантический анализ и генерация исходного кода.

В соответствии с приведенным выше тезисами были поставлены цели курсового проекта:

* разработать спецификацию языка программирования KVV-2024
* разработать структуру транслятора
* разработать структуру лексического анализатора
* разработать структуру синтаксического анализатора
* разработать структуру семантического анализатора
* разработать генерацию объектного кода
* составить контрольный пример для тестирования работы транслятора
* провести тестирование

Решения каждой из поставленных целей будут приведены в соответствующих разделах курсового проекта.

# Спецификация языка программирования

## 1.1. Характеристика языка программирования

Разрабатываемый язык программирования KVV-2024 – компилируемый, высокоуровневый, императивный, строго типизированный, явный.

## 1.2. Определение алфавита языка программирования

Язык программирования KVV-2024 использует стандартную кодировку символов Windows-1251. Его структура представлена в форме Бэкуса-Наура в таблице 1.1.

Таблица 1.1 – Алфавит языка программирования KVV-20K24

|  |
| --- |
| <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 |
| <символ- сепаратор>::= ' '|,|(|)|{|}|; |
| <побитовые операция>::= | | & | ! |

## 1.3. Применяемые сепараторы

Сепараторы являются неотъемлемой частью языка программирования. Они используются для логического разделения конструкций, функций и лексем.

Сепараторы, используемые в языке программирования KVV-2024, и их назначение представлены в таблице 1.2.

Таблица 1.2 – Применяемые сепараторы KVV-2024

|  |  |
| --- | --- |
| Сепаратор | Назначение |
| пробел | Разделитель лексем. |
| ; (точка с запятой) | Разделитель строк |
| {} | Определитель блока функций |
| () | Определитель параметров функций |
| , (запятая) | Разделитель параметров функции |
| ‘’ | Определение символьного литерала |

## 1.4. Применяемые кодировки

Для написания исходного кода программы на языке KVV-2024 используется кодировка Windows-1251. Русские символы разрешается использоваться только в символьных литералах.

## 1.5. Типы данных

Язык программирования KVV-2024 поддерживает использование следующих типов данных: беззнаковый целый, символьный. Описание типов данных представлено в таблице 1.3.

Таблица 1.3. – Типы данных KVV-2024

|  |  |
| --- | --- |
| Тип данных | Описание |
| Беззнаковый целый | Беззнаковый целочисленный тип данных. Занимает 4 байта. Диапазон значений: от 0 до 4294967295. Инициализация по умолчанию: 0. |
| Символьный | Символьный тип данных. символ занимает 1 байт. Диапазон значений: все символы кодировки Windows-1251. Инициализация по умолчанию: ‘\0’ (нулевой байт). |

## 1.6. Преобразование типов данных

Все типы данных определены однозначно и не могут быть преобразованы в другие.

## 1.7. Идентификаторы

Идентификатор языка – это последовательность символов, используемая в качестве имени для идентификации сущностей в языке программирования.

В KVV-2024 идентификаторы используются для именования переменных, функций и их параметров.

Формальное описание идентификатора представлено в форме Бэкуса-Наура в таблице 1.4

Таблица 1.4 – Идентификаторы KVV-2024

|  |
| --- |
| <идентификатор> ::= <буква> | '\_' {<буква> | <цифра> | '\_'} |
| <буква> ::= 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 | 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 |

В языке KVV-2024 идентификаторы не могут содержать символов кириллицы, специальных символов за исключением нижнего подчеркивания; не могут начинаться с цифры.

Примеры правильных идентификаторов: s0mthing, name, can\_get.

Примеры неправильных идентификаторов: 0var, somth!ng.

## 1.8. Литералы

Литералы языка программирования – последовательность символов, используемая для именования неизменяемых значений в коде.

В языке KVV-2024 предусмотрены следующие типы литералов: целочисленный, символьный. Описание литералов представлено в таблице 1.5.

Таблица 1.5. – Литералы KVV-2024

|  |  |
| --- | --- |
| Литерал | Описание |
| Беззнаковый целый | Могут состоять только из цифр[0..9](или в любых других системах исчисления) без дробной части и без знака.  При выходе за пределы допустимости будет выведена соответствующая ошибка.  Произвольная система счисления |
| Строковый | Состоит из символов, заключенных в ‘’ (одинарные кавычки). |
| Логический | Имеет только два значения: true, false |

*Пример правильных литералов: 2024, ‘Take your time’.*

*Пример неправильных литералов: help, -4.*

## 1.9. Объявление данных

При объявлении переменных в языке KVV-2024 необходимо учитывать следующие правила:

* каждая переменная должна быть объявлена до её использования;
* синтаксис: declare <тип данных> <идентификатор>;
* объявление начинается с ключевого слова declare, за которым следует идентификатор переменной и её тип;
* тип переменной должен быть одним из допустимых в KVV-2024 типов.

Определение функции схоже с объявлением переменной:

* каждая функция должна быть определена перед применением;
* синтаксис: <тип\_возвращаемого\_значения> function <имя\_функции> (<параметры>);
* определение функции начинается с типа возвращаемого значения, за которым следует ключевое слово function, имя функции, параметры в круглых скобках (или пустые скобки, если параметров нет) и тело функции в фигурных скобках.

## 1.10. Инициализация данных

Во время объявления переменной можно присвоить ей значение с помощью оператора присваивания (=). По умолчанию, переменным различных типов данных присваиваются начальные значения.

## 1.11. Инструкции языка

Инструкции языка программирования KVV-2024 представлены в таблице 1.6.

Таблица 1.6 – Инструкции языка KVV-2024

|  |  |  |
| --- | --- | --- |
| Инструкция | Описание | Пример |
| Объявление переменной | declare <тип данных> <идентификатор>; | declare integer number;  declare char symbol; |
| Инициализация переменной | <тип данных> <идентификатор> = <литерал> | <идентификатор> | declare integer number = 14;  declare char symbol = ‘c’; |
| Объявление фукнции | <тип данных> function <идентификатор функции> (<список параметров>) {<блок функции>}; | integer function sum(integer a, integer b) {}; |
| Выход из функции | return <идентификатор> | <литерал>; | return number;  return ‘V’; |
| Присваивание | <идентификатор> = <литерал>;  <идентификатор> = <идентификатор>; | number = 625;  number = symbol; |
| Вывод данных | write <идентификатор> | <литерал>; | write number; |
| Ввод данных | read <идентификатор>; | read number; |

## 1.12. Операции языка

Язык программирования KVV-2024 поддерживает различные типы операций. Их описание, приоритетность выполнения представлены в таблице 1.7.

Таблица 1.7 – Операции KVV-2024

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Операция | Арифметическое назначение | Приоритетность операции | Свойства | Количество операндов | Тип операндов | Тип результата | Порядок выполнения |
| | | Побитовое или | 3 | Коммутативность, дистрибутивность | 2 | Целочисленные беззнаковые | Целочисленный беззнаковый | Слева-направо |

Продолжение таблицы 1.7

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| & | Побитовое и | 2 | Коммутативность, дистрибутивность | 2 | Целочисленные беззнаковые | Целочисленный беззнаковый | Слева-направо |
| ! | Побитовая инверсия | 1 | Ассоциативность, коммутативность, дистрибутивность | 1 | Целочисленные беззнаковые | Целочисленный беззнаковый | Слева-направо |
| ^ | Исключающее ИЛИ | 2 | Коммутативность | 2 | Целочисленные беззнаковые | Целочисленный беззнаковый | Слева-направо |

## 1.13. Выражения и их вычисление

Выражение в KVVs-2024 — это комбинация литералов, переменных и операций, которые вместе дают некоторое значение.

Предусмотрены следующие правила составления выражений:

* Рассматриваются слева направо;
* Для изменения приоритета операции используются круглые скобки ();
* Каждое выражение должно заканчиваться точкой с запятой;
* Поддерживаются целочисленные выражения (все операции возвращают целое значение).

## 1.14 Конструкции языка

Конструкции языка KVV-2024 представлены в таблице 1.7.

Таблица 1.8 - Конструкции языка

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

Исходный текст программы на языке программирования KVV-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

Область видимости в языке KVV-2024 организована следующим образом: все идентификаторы доступны из текущей области видимости или из вложенных областей видимости. Области видимости выделяются {}.

## 1.16 Семантические проверки

В языке программирования KVV-2024 выполняются следующие семантические проверки:

* единственность точки входа;
* переопределение идентификаторов;
* использование идентификаторов без их объявления;
* проверка соответствия типа функции и возвращаемого параметра;
* правильность передаваемых в функцию параметров: количество, типы;
* правильность выражений;

## 1.17 Распределение оперативной памяти на этапе выполнения

Все переменные размещаются в стеке.

## 1.18 Стандартная библиотека и ее состав

Таблица 1.9 - Стандартная библиотека языка KVV-2024

|  |  |
| --- | --- |
| Функция | Описание |
| bool strEq (char\* a, char\* b); | Сравнивает две строки и возвращает результат их сравнения (равны, не равны).  Принимает два строковых параметра.  Возвращает логическое значение. |
| char\* findWord(char\* a, unsigned int n); | Находит слово в строке.  Принимает строковый и целочисленный параметр.  Возвращает строковое значение. |

## 1.19. Ввод и вывод данных

В языке KVV-2024 предусмотрен ввод данных, который осуществляется с помощью оператора read. В качестве аргумента которого, могут выступать идентификаторы, которым будет присвоено значение.

Вывод в данном языке программирования предусмотрен с помощью оператора write. В качестве аргумента выступает идентификатор, литерал и функция значение которых в дальнейшем будет выведено.

## 1.20 Точка входа

Точкой входа в программе является ключевое слово main. Точка входа не может отсутствовать. Но более чем одной точки входа быть не может.

## 1.21 Препроцессор

Препроцессор убирает комментарии.

## 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 вызываемым кодом.

## 1.23 Объектный код

Программа, написанная на языке программирования KVV-2024, будет транслироваться в ассемблерный код Intel x86.

## 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 - 59 | Системные ошибки |
| 60 - 99 | Лексические ошибки |
| 100 - 109 | Ошибки параметров |
| 110 - 119 | Ошибки открытия файлов |
| 120 - 139 | Синтаксические ошибки |
| 140 - 159 | Семантические ошибки |

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

## 1.25 Контрольный пример

Контрольный пример на языке KVV-2024 представлен в приложении А.

# Структура транслятора

## 2.1 Компоненты транслятора, их назначение и принципы взаимодействия

Транслятор — это программа, преобразующая исходный код с одного языка программирования на другой. Основные этапы его работы включают анализ исходного кода, его трансформацию и генерацию выходного кода.

В языке программирования KVV-2024 исходный код транслируется в ассемблерный код.

Графическое отображение структуры транслятора представлено на рисунке 2.1.

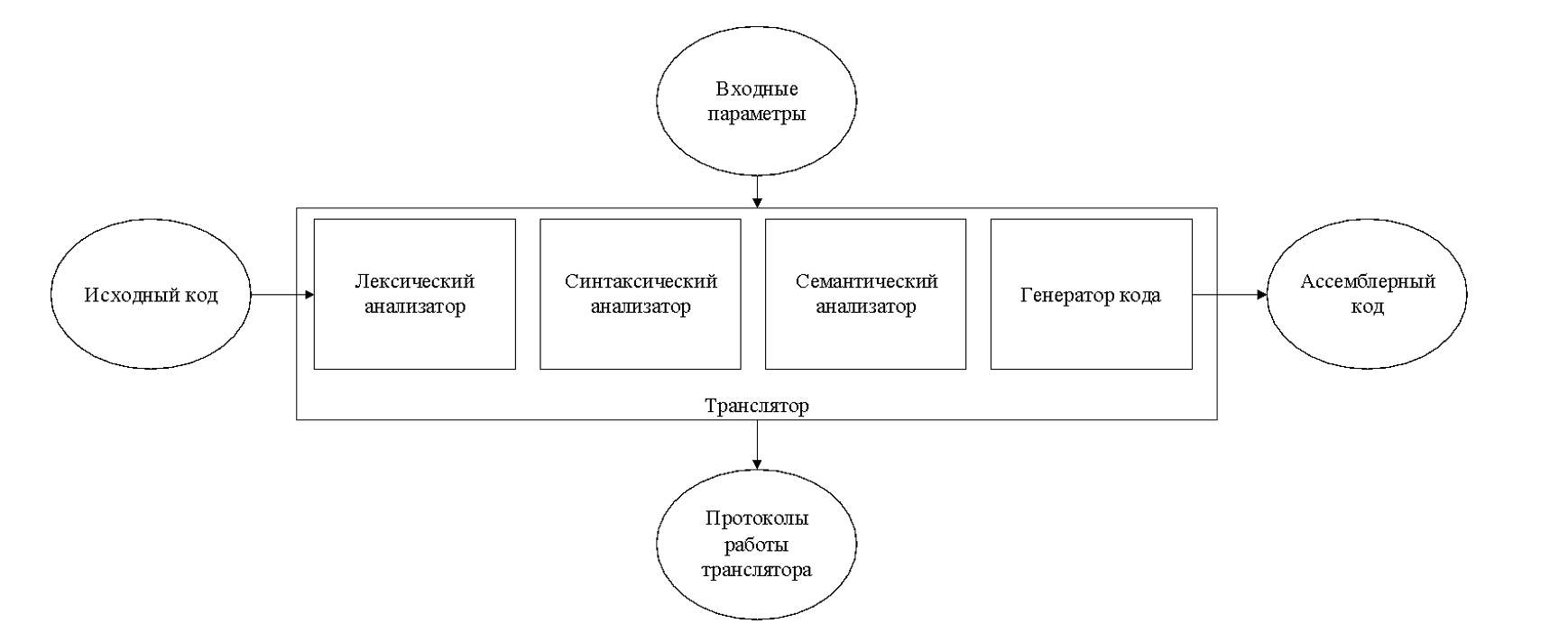


Рисунок 2.1 – Графическое отображение структуры транслятора

Лексический анализатор преобразует исходный текст в лексемы, синтаксический анализатор проверяет их синтаксическую корректность и строит дерево синтаксического разбора, семантический анализатор проверяет логическую корректность и добавляет семантическую информацию. Промежуточное представление используется для оптимизации кода, после чего генератор кода преобразует его в конечный выходной код. Эти этапы совместно обеспечивают полное преобразование исходного кода в готовый к выполнению код.

## 2.2 Перечень параметров транслятора

Входные параметры представлены в таблице 2.1.

Таблица 2.1 – Входные параметры транслятора языка KVV-2024

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

Входные параметры необходимы для формирования файлов, содержащих информацию о результате работы транслятора.

## 2.3 Протоколы, формируемые транслятором

Во время работы транслятора на каждом этапе, формируются рабочие протоколы. В таблице 2.2 приведен список протоколов, генерируемых транслятором, а также их содержимое.

Таблица 2.2 – Протоколы, создаваемые транслятором языка KVV-2024

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |
| Файл журнала, указанный параметром “-log:” | В этот файл включается информация о входных параметрах приложения, проверке допустимости символов, а также результаты работы лексического и семантического анализаторов. В случае возникновения ошибок на любом из этапов, информация об ошибках будет записана в этот файл. |
| Выходной файл с расширением “.asm” | Этот файл содержит исходный код на языке ассемблера и является результатом работы программы. |

Эти протоколы позволяют контролировать корректность, надежность и качество процесса трансляции, а также выявлять проблемы.

# Разработка лексического анализатора

## 3.1 Структура лексического анализатора

Первым шагом в преобразовании исходного кода является лексический анализ, выполняемый лексическим анализатором. Лексический анализатор — это часть компилятора, которая читает исходную программу и выделяет в ней лексемы. Лексема — это элементарная единица языка, состоящая из базовых символов и не включающая других структурных элементов. В языках программирования лексемами могут быть идентификаторы, константы, ключевые слова, операционные знаки и т.п. Структура лексического анализатора представлена на рисунке 3.1.

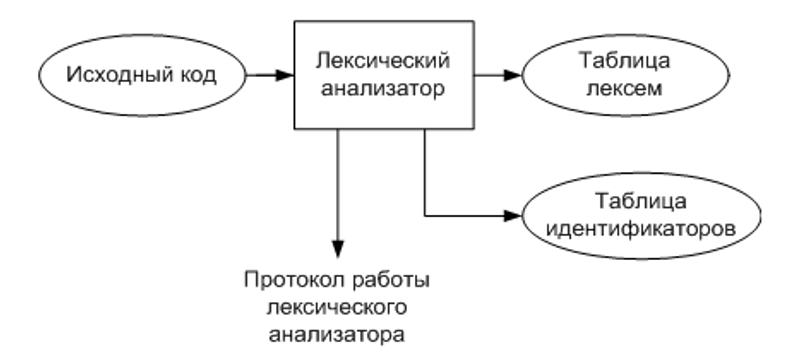


Рисунок 3.1 – Структура лексического анализатора

Входными данными для лексического анализатора является исходный код — текст программы, подлежащей анализу. В процессе анализа создаются две таблицы: таблица лексем, содержащая преобразованные значимые последовательности символов, и таблица идентификаторов, в которой хранятся идентификаторы и дополнительная информация о них. Также формируется протокол работы лексического анализатора, документирующий процесс анализа и возникающие ошибки. Управление работой анализатора осуществляется с помощью параметров, конфигурационных настроек, уровней логирования и режимов отладки.

## 3.2 Контроль входных символов

Код, написанный на языке программирования KVV-2024, служит входными данными. Чтобы правильно разбить исходный текст на слова, символы из таблицы кодировки Windows-1251 разделяются на категории. Таблица допустимости представлена на рисунке 3.2.

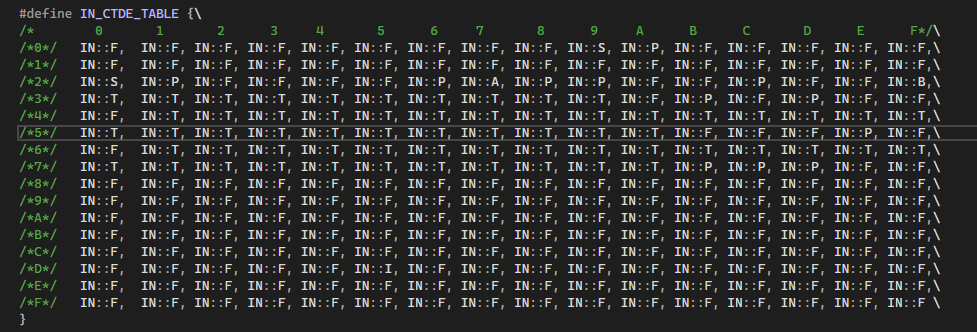


Рисунок 3.2 – Таблица допустимости входных символов

Эта таблица была сформирована на основе кодировки Windows-1251 и необходима для проверки допустимости символов. В таблице 3.1 приводятся символы и их значения.

Таблица 3.1 – Соответствие символов и их значений

|  |  |
| --- | --- |
| Символ | Значение |
| T | Разрешенный символ |
| F | Запрещенный символ |
| S | Символы-сепараторы |
| I | Игнорируемый символ |
| B | Символ начала комментария |
| A | Символ строкового литерала |
| P | Специальный символ |

## 3.3 Удаление избыточных символов

Избыточный символ — это любой символ в исходном коде программы, который не влияет на его логику и выполнение. Такие символы включают пробелы, табуляции, комментарии и другие, не влияющие на синтаксическую и семантическую корректность кода.

Описание алгоритма удаления избыточных символов:

* получение исходного текста программы;
* последовательное сканирование текста программы;
* проверка символов: при нахождении лишних пробелов или знака табуляции переходим к следующему символу;
* формирование результирующего текста, состоящего только из значимых символов;
* вывод результирующего текста, готового для дальнейшей обработки.

## 3.4 Перечень ключевых слов

Перечень ключевых слов и лексем приведены в таблице 3.2.

Таблица 3.2 – Перечень ключевых слов и лексем

|  |  |  |  |
| --- | --- | --- | --- |
| Тип цепочки | Назначение | Цепочка | Лексема |
| Тип данных | Представление целочисленных данных (1 байт) | integer | t |
| Представление символьных данных | char |
| Лексема | Объявление переменной | declare | d |
| Объявление функции | function | f |
| Возврат значения из функции | return | r |
| Условный оператор | if | ? |
| Ввод данных | read | e |
| Вывод данных | write | p |
| Блок функции | { | { |
| } | } |
| Оператор | Oперации | | | | |
| & | & |
| ! | ! |
| ^ | ^ |
| Оператор присваивания | = | = |
| Идентификатор | - | [a-z;A-Z]+  [a-z;A-Z;0-9]\* | i |
| Точка входа | - | main | m |
| Литерал | Целочисленный | [1-9]+[0-9]\*  0b[0-1]\*  0[0-7]\*  0x[1-F]+[0-F]\* | l |
| Строковый | [a-z;A-Z]+  [a-z;A-Z;0-9]\* |

В приложении А находятся конечные автоматы, соответствующие лексемам языка KVV-2024.

## 3.5 Основные структуры данных

Основные структуры данных на этапе лексического анализа представлены в листинге 3.1 и 3.2.

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.

|  |
| --- |
| struct Entry  {  int idxfirstLE;  char\* id;  IDDATATYPE iddatatype;  IDTYPE idtype;  std::stack<char\*> funcID;  union  {  int vint;  struct  {  int len;  char\* str;  } vstr;  } value;  };  struct IdTable  {  int maxsize;  int size;  Entry\* table;  }; |

Листинг 3.1 – Структура таблицы идентификаторов

|  |
| --- |
| struct Entry  {  char lexema;  int sn;  int idxTI;  };  struct LexTable  {  int maxsize;  int size;  Entry\* table;  }; |

Листинг 3.2 – Структура таблицы лексем

## 3.6 Структура и перечень сообщений лексического анализатора

Индексы ошибок, обнаруживаемых лексическим анализатором, находятся в диапазоне 90-99. Перечень сообщений лексического анализатора представлен в листинге 3.3.

|  |
| --- |
| ERROR\_ENTRY(90, "Лексический анализатор: слово не распознано"),  ERROR\_ENTRY(91, "Лексический анализатор: строковый литерал превышает допустимый размер"),  ERROR\_ENTRY(92, "Лексический анализатор: строковый литерал не распознан"),  ERROR\_ENTRY(93, "Лексический анализатор: идентификатор не распознан"),  ERROR\_ENTRY(94, "Лексический анализатор: найдено несколько функций main"),  ERROR\_ENTRY(95, "Лексический анализатор: функция объявлена несколько раз"),  ERROR\_ENTRY(96, "Лексический анализатор: переменная объявлена несколько раз"),  ERROR\_ENTRY(97, "Лексический анализатор: неизвестная переменная"),  ERROR\_ENTRY(98, "Лексический анализатор: не найдена точка входа(main)"),  ERROR\_ENTRY(99, "Лексический анализатор: ключевое слово используется как идентификатор"), |

Листинг 3.3 – Перечень ошибок лексического анализатора

## 3.7 Принцип обработки ошибок

Если транслятор обнаруживает ошибки, он прекращает выполнение и записывает соответствующие сообщения об ошибках в log-файл. Сообщения содержат информацию о строке и позиции, где была выявлена ошибка.

## 3.8 Параметры лексического анализатора

Параметры, управляющие работой лексического анализатора, представлены в таблице 3.3.

Таблица 3.3 – Параметры лексического анализатора

|  |  |  |
| --- | --- | --- |
| 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. |

Продолжение таблицы 3.3

|  |  |  |
| --- | --- | --- |
| Таблица допустимости входных символов | Определяет, какие символы считаются допустимыми в исходном коде. | Лексический анализатор различает символы, определенные в наборе, и игнорирует остальные. |
| Таблица лексем | Используется для хранения информации о ранее обработанных лексемах. | Лексический анализатор использует таблицу символов для сохранения и поиска информации о символах в коде. |

## 3.9 Алгоритм лексического анализа

Последовательность выполнения алгоритма работы лексического анализатора:

* установление начальных параметров, таких как текущая позиция в исходном коде;
* постепенное считывание символов из входного потока;
* для каждого прочитанного символа определение его типа (буква, цифра, пробел и т.д.);
* сборка символов в лексемы (токены) в зависимости от их типа и правил грамматики;
* определение типа каждой лексемы (ключевое слово, идентификатор, число и т.д.);
* если встречается некорректный символ или последовательность символов, формирование сообщения об ошибке;
* возвращение готовых лексем синтаксическому анализатору;
* когда весь исходный код обработан, передача специальных лексем, указывающих на конец входного потока.

## 3.10 Контрольный пример

Результат работы лексического анализатора – таблицы лексем и идентификаторов представлены в Приложении Б.

# Разработка синтаксического анализатора

## 4.1 Структура синтаксического анализатора

Синтаксический анализ — это этап трансляции, следующий за лексическим анализом и предназначенный для распознавания синтаксических конструкций. Структура синтаксического анализатора представлена на рисунке 4.1.

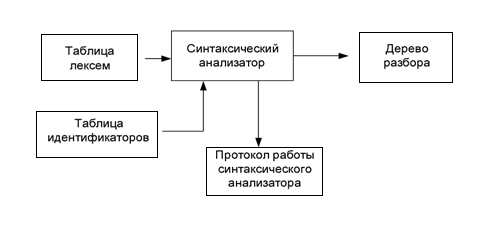


Рисунок 4.1 – Структура синтаксического анализатора

Входными данными для синтаксического анализа являются таблица лексем и таблица идентификаторов. Также создается протокол, в котором документируется процесс анализа и фиксируются возникающие ошибки. В результате работы синтаксического анализатора формируется дерево разбора.

## 4.2 Контекстно-свободная грамматика, описывающая синтаксис языка

В синтаксическом анализаторе транслятора языка KVV-2024 используется контекстно-свободная грамматика , где

T – множество терминальных символов (было описано в разделе 1.2 данной пояснительной записки),

N – множество нетерминальных символов (первый столбец таблицы 4.1),

P – множество правил языка (второй столбец таблицы 4.1),

S – начальный символ грамматики, являющийся не терминалом.

Эта грамматика имеет нормальную форму Грейбах, т.к. она не леворекурсивная (не содержит леворекурсивных правил) и правила  имеют вид:

1. , где ; (или , или )
2. , где – начальный символ, при этом если такое правило существует, то не терминал  не встречается в правой части правил.

Грамматика языка KVV-2024 представлена в приложении Б.

TS – терминальные символы, которыми являются сепараторы, знаки арифметических операций и некоторые строчные буквы.

NS – нетерминальные символы, представленные несколькими заглавными буквами латинского алфавита.

Перечень правил, составляющих грамматику языка представлен в таблице 4.1.

Таблица 4.1 – Перечень правил, составляющих грамматику языка и описание нетерминальных символов KVV-2024

|  |  |  |
| --- | --- | --- |
| Не терминал | Цепочки правил | Описание |
| S | m{NrE;}S  m{NrE;}  m{rE;}S  m{rE;}  tfi(F){NrE;}S  tfi(F){NrE;}  tfi(F){rE;}S  tfi(F){rE;}  tfi(){NrE;}S  tfi(){NrE;}  tfi(){rE;}S  tfi(){rE;} | Порождает правила, описывающее общую структуру программы |
| N | dti;  dti=E;  i=E;  rE;  dti;N  dti=E;N  ri;N  rl;N  i=E;N  w(E);  w(E);N  e(i);  e(i);N  ?(E){N}  ?(E){N}N | Порождает правила, описывающие конструкции языка |
| E | i  l  (E)  i(W)  i()  iM  lM  (E)M  I(W)M | Порождает правила, описывающие выражения |

Продолжение таблицы таблицы 4.1

|  |  |  |
| --- | --- | --- |
| W | i  l  i,W  l,W | Порождает правила, описывающие принимаемые параметры функции |
| F | ti  ti,F | Порождает правила, описывающие параметры локальной функции при её объявлении |
| M | |E  |EM  &E  &EM  !E  !EM  ^E  ^EM | Порождает правила, описывающие знаки побитовых операций (|,&,!,^) |

## 4.3 Построение конечного магазинного автомата

Конечный автомат с магазинной памятью представляет собой семерку, описание которой представлено в таблице 4.2. Структура данного автомата показана в Приложении В.

Таблица 4.2 – Описание компонентов магазинного автомата

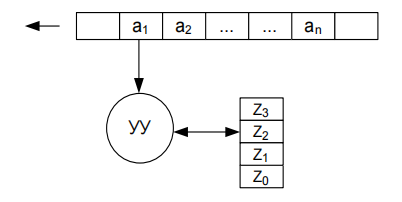
|  |  |  |
| --- | --- | --- |
| Компонента | Определение | Описание |
|  | Множество состояний автомата | Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата |
|  | Алфавит входных символов | Алфавит является множеством терминальных и нетерминальных символов, описание которых содержится в разделе 1.2 и в таблице 4.1. |
|  | Алфавит специальных магазинных символов | Алфавит магазинных символов содержит стартовый символ и маркер дна стека |
|  | Функция переходов автомата | Функция представляет из себя множество правил грамматики, описанных в таблице 4.1. |
|  | Начальное состояние автомата | Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики (нетерминальный символ А) |

Продолжение таблицы таблицы 4.2

|  |  |  |
| --- | --- | --- |
|  | Начальное состояние магазина автомата | Символ маркера дна стека ($) |
|  | Множество конечных состояний | Конечные состояние заставляют автомат прекратить свою работу. Конечным состоянием является пустой магазин автомата и совпадение позиции на входной ленте автомата с размером ленты |

Протокол и ошибки работы синтаксического анализатора выводятся в лог журнал.

Схема работы конечного автомата с магазинной памятью представлена на рисунке 4.2.

Рисунок 4.2 – Схема работы конечного автомата с магазинной памятью

## 4.4 Основные структуры данных

Основные структуры данных синтаксического анализатора включают в себя структуру магазинного конечного автомата и структуру грамматики Грейбах, описывающей правила языка KVV-2024. Данные структуры представлены в Приложении В.

## 4.5 Описание алгоритма синтаксического разбора

Принцип работы автомата:

* в магазин записывается стартовый символ;
* на основе полученных ранее таблиц формируется входная лента;
* запускается автомат;
* выбирается цепочка, соответствующая нетерминальному символу, записывается в магазин в обратном порядке;
* если терминалы в стеке и в ленте совпадают, то данный терминал удаляется из ленты и стека. Иначе возвращаемся в предыдущее сохраненное состояние и выбираем другую цепочку не терминала;
* если в магазине встретился не терминал, переходим к пункту 4;
* если наш символ достиг дна стека, и лента в этот момент пуста, то синтаксический анализ выполнен успешно и формируется дерево разбора. Иначе генерируется исключение.

## 4.6 Структура и перечень сообщений синтаксического анализатора

Индексы ошибок, обнаруживаемых синтаксическим анализатором, находятся в диапазоне 130-134. Перечень сообщений синтаксического анализатора представлен в листинге 4.1.

|  |
| --- |
| ERROR\_ENTRY(130,"Синтаксический анализатор: Неправильная структура программы"),  ERROR\_ENTRY(131,"Синтаксический анализатор: Неправильный идентификатор"),  ERROR\_ENTRY(132,"Синтаксический анализатор: Произошла ошибка в параметрах функции"),  ERROR\_ENTRY(133,"Синтаксический анализатор: Произошла ошибка в параметрах функции, которая в данный момент вызывается."),  ERROR\_ENTRY(134,"Синтаксический анализатор: Формула в параметрах вычисляемой функции"), |

Листинг 4.1 – Перечень сообщений синтаксического анализатора

## 4.7 Параметры синтаксического анализатора и режимы его работы

Синтаксический анализатор принимает на вход таблицу лексем и идентификаторов, а также использует описание грамматики в форме Грейбах. Результатом его работы являются дерево разбора и протокол работы автомата с магазинной памятью, которые записываются в журнал программы.

## 4.8 Принцип обработки ошибок

Процесс обработки ошибок включает следующие этапы:

* синтаксический анализатор проверяет все правила и цепочки правил грамматики, чтобы найти соответствие с конструкцией, представленной в таблице лексем;
* если не удается найти подходящую цепочку, генерируется соответствующее сообщение об ошибке;
* при возникновении ошибки сообщение об ошибке записывается в журнал логов, после чего компилятор прекращает работу.

## 4.9 Контрольный пример

Пример разбора синтаксическим анализатором исходного кода на языке KVV-2024 представлен в Приложении Д. Дерево разбора исходного кода приложен к проекту.

# Разработка семантического анализатора

## 5.1 Структура семантического анализатора

Семантический анализ является третьей фазой работы транслятора. Он реализуется в виде отдельных проверок различных ситуаций в конкретных случаях, таких как установка флага, нахождение в особом месте программы (например, оператор выхода из функции, оператор ветвления или вызов функции стандартной библиотеки). Структура семантического анализатора представлена на рисунке 5.1.

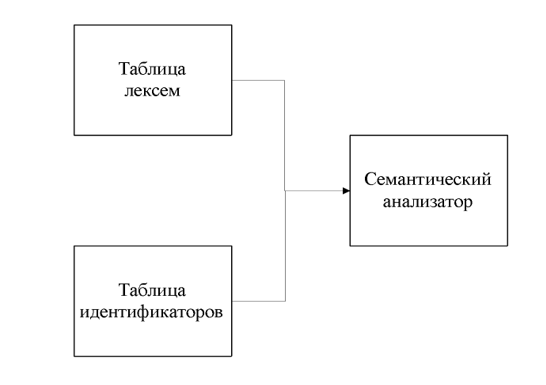


Рисунок 5.1 – Структура семантического анализатора

Семантический анализатор включает ряд функций для проверки корректности исходной программы. Эти функции выполняются на различных этапах работы транслятора.

## 5.2 Функции семантического анализатора

Семантический анализатор осуществляет проверку на соответствие основным правилам языка, которые описаны в разделе 1.16. Эти правила включают в себя контроль над правильностью использования переменных, функций, операторов и других языковых конструкций, обеспечивая корректность и целостность исходного кода.

## 5.3 Структура и перечень сообщений семантического анализатора

Сообщения, формируемые семантическим анализатором, представлены в листинге 5.1.

|  |
| --- |
| ERROR\_ENTRY(140, "Семантический анализатор: Неверные параметры функции"),  ERROR\_ENTRY(141, "Семантический анализатор: Неверный тип данных") |

Листинг 5.1 – Сообщения, формируемые семантическим анализатором

## 5.4 Принцип обработки ошибок

Ошибки, возникающие в процессе трансляции программы, записываются в протокол, указанный во входных параметрах. При обнаружении ошибок они фиксируются в протоколе. Все семантические ошибки считаются критическими, поэтому при их возникновении транслятор останавливает свою работу.

## 5.5 Контрольный пример

При нахождении ошибок в коде на языке программирования KVV-2024 программа прекращает свое действие и выводит сообщение об ошибке. Контрольный пример с демонстрацией ошибки, диагностируемой семантическим анализатором представлен в листинге 5.2.

|  |
| --- |
| integer function fi(integer x, integer y)  {  declare integer z;//я люлю коменты  z = x & (x|y);  return z;  }  main  {  declare integer z;  z = fi(x,z,b);  return 0;  }  Ошибка 141: Семантический анализатор: Неверный тип данных  Cтрока: 10 |

Листинг 5.2 – Контрольный пример

# Вычисление выражений

## 6.1 Выражения, допускаемые языком

В языке KVV-2024 допускаются выражения, применимые к целочисленным типам данных. В выражениях поддерживаются побитовые операции, такие как: &, |, ^, ! и (). А также вызовы функций как операнды арифметических выражений. Приоритет операций представлен в таблице 6.1.

**6.2 Польская запись и принцип ее построения**

Выражения в языке KVV-2024 преобразовываются к обратной польской записи.

Польская запись – это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок.

Обратная польская запись – это форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. 

Алгоритм построения:

– читаем очередной символ;

– если он является идентификатором или литералом, то добавляем его к выходной строке;

– если символ является символом функции, то помещаем его в стек;

– если символ является открывающей скобкой, то она помещается в стек;

– исходная строка просматривается слева направо;

– если символ является закрывающей скобкой, то выталкиваем из стека в выходную строку все символы пока не встретим открывающую скобку. При этом обе скобки удаляются и не попадают в выходную строку;

– как только входная лента закончится все символы из стека выталкиваются в выходную строку;

– в случае если встречаются операции, то выталкиваем из стека в выходную строку все операции, которые имеют выше приоритетность чем последняя операция;

Пример преобразования выражения в обратную польскую запись представлен в таблице 6.1.

Таблица 6.1 – Пример преобразования выражение в обратную польскую запись

|  |  |  |
| --- | --- | --- |
| Исходная строка | Результирующая строка | Стек |
| x&(x|y) |  |  |
| &(x|y) | x |  |
| (x|y) | x | & |
| x|y) | x | &( |
| |y) | xx | &( |
| y) | xx | &(| |

Продолжение таблицы таблицы 6.1

|  |  |  |
| --- | --- | --- |
| ) | xxy | &(| |
|  | xxy? | & |
|  | xxy?& |  |

**6.3 Программная реализация обработки выражений**

Программная реализация алгоритма преобразования выражений к польской записи представлена в Приложении Е.

**6.4 Контрольный пример**

Пример преобразования выражения к польской записи представлен в таблице 6.1. Преобразование выражений в формат польской записи необходимо для построения более простых алгоритмов их вычисления.

# Генерация кода

## 7.1 Структура генератора кода

Генерация ассемблерного кода – это перевод компилятором внутреннего представления исходной программы KVV-2024 в цепочку символов выходного языка. На вход генератора подаются таблицы лексем и идентификаторов, на основе которых генерируется файл с ассемблерным кодом.

Структура генератора кода представлена на рисунке 7.1.



Рисунок 7.1 – Структура генератора кода

## 7.2 Представление типов данных в оперативной памяти

Элементы таблицы идентификаторов расположены в разных сегментах языка ассемблера – .data и .const. Соответствия между типами данных идентификаторов на языке KVV-2024 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка KVV-2024 и языка Ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке KVV-2024 | Тип идентификатора на языке ассемблера | Пояснение |
| char | BYTE | Хранение символьного типа данных. Занимает 1 байт |
| integer | DWORD | Хранит целочисленный тип данных. Занимает 4 байта |
| Лексема | BYTE/DWORD | Литералы:  целочисленные, строковые |

Идентификаторы языка KVV-2024 размещены в сегменте данных (.data). Литералы – в сегменте констант (.const).

## 7.3 Статическая библиотека

В языке KVV-2024 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++. Объявление функций статической библиотеки генерируется автоматически в коде ассемблера.

В таблице 7.2 описываются функции, доступные в статической библиотеке языка KVV-2024. Эти функции могут быть полезны при работе с выводом на консоль, а также выполняют математические операции.

Таблица 7.2 – Функции, доступные в статической библиотеке языка KVV-2024

|  |  |
| --- | --- |
| Функции | Назначение |
| void OutputInt(int num) | Выводит число на экран |
| void OutputChar(char\* ptr) | Выводит строку на экран |
| void rusL() | Включает поддержку русского языка. Вызывается автоматически |
| int InputInt() | Ввод числа |
| int strEq(char\* a, char\* b) | Сравнение строк |
| char\* findWord(char\* a, unsigned int n) | Находит слово в строке |

**7.4 Особенности алгоритма генерации кода**

Алгоритм генерации кода выглядит следующим образом:

1. Генерирует заголовочную информацию (Листинг 7.1): модель памяти, подключение библиотек, прототипы внешних функций, размер стека.

;========================= шапка ==============================

.586

.model flat, stdcall

includelib libucrt.lib

includelib kernel32.lib

includelib standartLib.lib

ExitProcess PROTO : DWORD

EXTRN rusL: proc

EXTRN OutputInt: proc

EXTRN OutputChar: proc

EXTRN InputInt: proc

EXTRN strEq: proc

EXTRN findWord: proc

.stack 4096

Листинг 7.1 – Пример заголовочной информации

1. Проходит полностью таблицу идентификаторов и заполняет поле .const литералами. Результат представлен в листинге 7.2.

;=================== сегмент констант =========================

.const

INT00GLOBAL DWORD 45

BOOl10GLOBAL BYTE 0

INT20GLOBAL DWORD 39

STR30GLOBAL BYTE '1234567890', 0

Продолжение листинга 7.2

STR40GLOBAL BYTE 'контрольный пример', 0

BOOl50GLOBAL BYTE 1

INT60GLOBAL DWORD 0

Листинг 7.2 – Пример заполнения поля .const

1. Проходим таблицу идентификаторов и объявляем переменные в поле .data. Результат заполнения поля .data представлен в листинге 7.3.

;=================== сегмент данных ============================

.data

buffer BYTE 256 dup(0)

zfi0GLOBAL DWORD ?

xMAIN DWORD ?

zMAIN DWORD ?

saMAIN DWORD ?

bMAIN DWORD ?

Листинг 7.3 – Пример заполнения поля .data

4) Генерируем сегмент данных .code. Сперва проходим по таблице идентификаторов и ищем функции. Объявляем их и генерируем код, содержащийся в функциях. Пример сгенерированной функции представлен в листинге 7.4.

;=================== сегмент кода ==============================

.code

fi0GLOBAL PROC xfi0GLOBAL : DWORD, yfi0GLOBAL : DWORD

push xfi0GLOBAL

push xfi0GLOBAL

push yfi0GLOBAL

pop eax

pop ebx

or eax, ebx

push eax

pop eax

pop ebx

and eax, ebx

push eax

pop zfi0GLOBAL

push strEq

jmp local0

local0:

pop eax

ret

fi0GLOBAL ENDP

main PROC

call rusL

push INT00GLOBAL

pop zMAIN

mov al, BOOl10GLOBAL

push ax

pop bMAIN

Продолжение листинга 7.4

push INT20GLOBAL

pop xMAIN

push offset STR30GLOBAL

pop saMAIN

push xMAIN

push zMAIN

call fi0GLOBAL

push eax

push fi0GLOBAL

pop zMAIN

push ecx

push offset STR40GLOBAL

call OutputChar

pop ecx

pop ecx

push ecx

call InputInt

mov zMAIN, eax

pop ecx

mov al, BOOl50GLOBAL

cmp al, 1

jz m0

jnz m1

je m0

m0:

push zMAIN

pop eax

not eax

push eax

pop zMAIN

push ecx

push zMAIN

call OutputInt

pop ecx

pop ecx

m1:

push strEq

jmp theend

theend:

call ExitProcess

main ENDP

end main

;=============================================================

Листинг 7.4 – Пример функции, полученной в результате генерации

**7.5 Входные параметры, управляющие генерацией кода**

На вход генератору кода поступают таблицы лексем и идентификаторов исходного кода программы на языке KVV-2024. Результаты работы генератора кода выводятся в файл с расширением .asm.

**7.6 Контрольный пример**

Генерируемый код записывается в файл «int.txt.asm». Сгенерированный код представлен в приложении Ж.

# Тестирование транслятора

**8.1 Общие положения**

Все тесты предназначены для выявления типичных пользовательских ошибок при использовании языка программирования. При обнаружении ошибки компилятор заносит её в протокол с номером ошибки и диагностическим сообщением, чтобы разработчик мог понять причину ошибки. После этого компилятор может продолжить анализ для обнаружения других возможных ошибок. Результаты тестирования записываются в лог-файл (.log).

Пример протокола, с выводом диагностики контрольного примера представлен в приложении З.

**8.2 Результаты тестирования**

В языке KVV-2024 не разрешается использовать запрещённые входным алфавитом символы где-либо кроме строковых или символьных переменных. Результат использования запрещённого символа показан в таблице 8.1.

Таблица 8.1 – Тестирование фазы проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Назначение |
| int | Ошибка 111: Недопустимый символ в исходном файле (-in)  Cтрока: 11, позиция: 11 |

На этапе лексического анализа могут возникнуть ошибки, описанные в пункте 3.6. Ошибка лексического анализатора приводит к прекращению выполнения программы и записи соответствующей ошибки в лог журнал.

Результаты тестирования лексического анализатора показаны в таблице 8.2.

Таблица 8.2 – Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Назначение |
| {} | Ошибка: 98: Лексический анализатор: не найдена точка входа(main) |
| main{…}main{…} | Ошибка 94: Лексический анализатор: найдено несколько функций main |
| z= x & (x | y); | Ошибка 97: Лексический анализатор: неизвестная переменная |

Продолжение таблицы 8.2

|  |  |
| --- | --- |
| Function int read (int a, int b) | Ошибка 99: Лексический анализатор: ключевое слово используется как идентификатор  Cтрока: 2 |

На этапе синтаксического анализа могут возникнуть ошибки, описанные в пункте 4.6. Результаты тестирования синтаксического анализатора показаны в таблице 8.3.

Таблица 8.3 – Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Исходный код | Назначение |
| fi();  main{  } | Ошибка 130 Синтаксический анализатор: Неправильная структура программы  Лексема: 'i', Строка 7 |
| int func fi (int x int y){} | Ошибка 134 Синтаксический анализатор: Формула в параметрах вычисляемой функции  Лексема: 't', Строка 2 |
| write; | Ошибка 132 Синтаксический анализатор: Произошла ошибка в параметрах функции  Лексема: ';', Строка 21 |

Итоги тестирования семантического анализатора приведены в таблице 8.4

Таблица 8.4 – Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Назначение |
| declare char sa = 01101; | Ошибка 141: Семантический анализатор: Неверный тип данных  Cтрока: 13 |
| z = fi(x,z,x); | Ошибка 140: Семантический анализатор: Неверные параметры функции  Cтрока: 16 |

**Заключение**

По окончании выполнения всех пунктов, изложенных ранее, получили рабочий транслятор языка программирования KVV-2024 на язык ассемблера.

Таким образом, были выполнены основные задачи данной курсовой работы:

* Сформулирована спецификация языка KVV-2024;
* Разработаны конечные автоматы и важные алгоритмы на их основе для эффективной работы лексического анализатора;
* Осуществлена программная реализация лексического анализатора, распознающего допустимые цепочки спроектированного языка;
* Разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
* Осуществлена программная реализация синтаксического анализатора;
* Разработан семантический анализатор, осуществляющий проверку используемых инструкций на соответствие логическим правилам;
* Разработан транслятор кода на язык ассемблера;
* Проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка KVV-2024 включает:

* 3 типа данных;
* Поддержка операторов ввода и вывода;
* Возможность вызова функций стандартной библиотеки;
* Наличие 4 побитовых операторов для вычисления выражений;
* Поддержка функций, операторов условия;
* Структурированная и классифицированная система для обработки ошибок пользователя.

**Список используемых источников**

1. Habr – Об изучении компиляторов и создании языков программирования [Электронный ресурс]. – Режим доступа: https://habr.com/ru/search/.

2. Построение компиляторов / Никлаус Вирт 2010. – 194 с.

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

4. Язык программирования C++.Лекции и упражнения [6-е издание] / Стивен Прата 2019 – 1094 с.

**Приложение А**

integer function fi(integer x, integer y)

{

declare integer z;//я люлю коменты

z= x & (x|y);

return z;

}

main

{

declare integer x;

declare integer z = 45;

declare char sa;

declare bool b;

b = false;

x = 047;

sa = '1234567890';

z = fi(x,z);

write 'контрольный пример';

read z;

if (true){

z = !z;

write z;

}

return 0;

}

Листинг А.1 - Контрольный пример

**Приложение Б**

--—------------------------------------------- Идентификаторы ---------------------------------------——-----------------

| Номер | id | Тип данных | Тип | Связь (Номер Лексема Строка) | Видимость | Значение ?

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

| 1 | strEq | bool | функция | 0 $ 0 | 0GLOBAL |

| 2 | a | str | параметр | 0 $ 0 | strEq -> 0GLOBAL |

| 3 | b | str | параметр | 0 $ 0 | strEq -> 0GLOBAL |

| 4 | findWord | str | функция | 0 $ 0 | 0GLOBAL |

| 5 | a | str | параметр | 0 $ 0 | findWord -> 0GLOBAL |

| 6 | n | int | параметр | 0 $ 0 | findWord -> 0GLOBAL |

| 7 | fi | int | функция | 3 i 1 | 0GLOBAL |

| 8 | x | int | параметр | 6 i 1 | fi -> 0GLOBAL |

| 9 | y | int | параметр | 9 i 1 | fi -> 0GLOBAL |

| 10 | z | int | переменная | 14 i 3 | fi -> 0GLOBAL |

| 11 | x | int | переменная | 34 i 9 | MAIN -> 0GLOBAL |

| 12 | z | int | переменная | 38 i 10 | MAIN -> 0GLOBAL |

| 13 | INT0 | int | литерал | 40 l 10 | 0GLOBAL | 45

| 14 | sa | str | переменная | 44 i 11 | MAIN -> 0GLOBAL |

| 15 | b | bool | переменная | 48 i 12 | MAIN -> 0GLOBAL |

| 16 | BOOl1 | bool | литерал | 52 l 13 | 0GLOBAL | 0

| 17 | INT2 | int | литерал | 56 l 14 | 0GLOBAL | 39

| 18 | STR3 | str | литерал | 60 l 15 | 0GLOBAL | 10 '1234567890'

| 19 | STR4 | str | литерал | 72 l 17 | 0GLOBAL | 20 'контрольный пример'

| 20 | BOOl5 | bool | литерал | 79 l 19 | 0GLOBAL | 1

| 21 | INT6 | int | литерал | 92 l 23 | 0GLOBAL | 0

Листинг Б.1 - Таблица идентификаторов

--—- Представления кода в виде лексем ——--

01 tfi|7|(ti|8|,ti|9|)

02 {

03 dti|10|;

04 i|10|=i|8|&(i|8||i|9|);

05 ri|10|;

06 }

07 m

08 {

09 dti|11|;

10 dti|12|=l;

11 dti|14|;

12 dti|15|;

13 i|15|=l;

14 i|11|=l;

15 i|14|=l;

16 i|12|=i|7|(i|11|,i|12|);

17 pl;

18 ni|12|;

19 ?(l){

20 i|12|=!i|12|;

21 pi|12|;

22 }

23 rl;

24 }

Листинг Б.2 – Таблица лексем

FST Ilex(

(char\*)word,

8,

NODE(1, RELATION('i', 1)),

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

NODE(1, RELATION('t', 3)),

NODE(1, RELATION('e', 4)),

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

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

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

NODE()

);

FST Slex(

(char\*)word,

5,

NODE(1, RELATION('c', 1)),

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

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

NODE(1, RELATION('r', 4)),

NODE()

);

FST Blex(

(char\*)word,

5,

NODE(1, RELATION('b', 1)),

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

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

NODE(1, RELATION('l', 4)),

NODE()

);

if (execute(Ilex) || execute(Slex) || execute(Blex))

return LEX\_INTEGER;

FST Flex(

(char\*)word,

9,

NODE(1, RELATION('f', 1)),

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

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

NODE(1, RELATION('c', 4)),

NODE(1, RELATION('t', 5)),

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

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

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

NODE());

if (execute(Flex))

return LEX\_FUNCTION;

FST Dlex(

(char\*)word,

8,

NODE(1, RELATION('d', 1)),

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

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

NODE(1, RELATION('l', 4)),

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

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

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

NODE()

);

if (execute(Dlex))

return LEX\_DECLARE;

FST Plex(

(char\*)word,

6,

NODE(1, RELATION('w', 1)),

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

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

NODE(1, RELATION('t', 4)),

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

NODE()

);

if (execute(Plex))

return LEX\_PRINT;

FST RElex(

(char\*)word,

5,

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

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

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

NODE(1, RELATION('d', 4)),

NODE()

);

if (execute(RElex))

return LEX\_READ;

FST Mlex(

(char\*)word,

5,

NODE(1, RELATION('m', 1)),

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

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

NODE(1, RELATION('n', 4)),

NODE()

);

if (execute(Mlex))

return LEX\_MAIN;

FST Rlex(

(char\*)word,

7,

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

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

NODE(1, RELATION('t', 3)),

NODE(1, RELATION('u', 4)),

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

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

NODE()

);

if (execute(Rlex))

return LEX\_RETURN;

FST IFlex(

(char\*)word,

3,

NODE(1, RELATION('i', 1)),

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

NODE()

);

FST Tlex(

(char\*)word,

5,

NODE(1, RELATION('t', 1)),

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

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

NODE(4, RELATION('e', 4)),

NODE()

);

FST Flex(

(char\*)word,

6,

NODE(1, RELATION('f', 1)),

NODE(2, RELATION('a', 2)),

NODE(3, RELATION('l', 3)),

NODE(4, RELATION('s', 4)),

NODE(5, RELATION('e', 5)),

NODE()

);

if (execute(Tlex) || execute(Flex))

return 'b';

FST strLit(

word,

4,

NODE(1, RELATION('\'', 1)),

NODE(258,

RELATION('A', 1), RELATION('A', 2), RELATION('B', 1), RELATION('B', 2),

RELATION('C', 1), RELATION('C', 2), RELATION('D', 1), RELATION('D', 2),

RELATION('E', 1), RELATION('E', 2), RELATION('F', 1), RELATION('F', 2),

RELATION('G', 1), RELATION('G', 2), RELATION('H', 1), RELATION('H', 2),

RELATION('I', 1), RELATION('I', 2), RELATION('J', 1), RELATION('J', 2),

RELATION('K', 1), RELATION('K', 2), RELATION('L', 1), RELATION('L', 2),

RELATION('M', 1), RELATION('M', 2), RELATION('N', 1), RELATION('N', 2),

RELATION('O', 1), RELATION('O', 2), RELATION('P', 1), RELATION('P', 2),

RELATION('Q', 1), RELATION('Q', 2), RELATION('R', 1), RELATION('R', 2),

RELATION('S', 1), RELATION('S', 2), RELATION('T', 1), RELATION('T', 2),

RELATION('U', 1), RELATION('U', 2), RELATION('V', 1), RELATION('V', 2),

RELATION('W', 1), RELATION('W', 2), RELATION('X', 1), RELATION('X', 2),

RELATION('Y', 1), RELATION('Y', 2), RELATION('Z', 1), RELATION('Z', 2),

RELATION('a', 1), RELATION('a', 2), RELATION('b', 1), RELATION('b', 2),

RELATION('c', 1), RELATION('c', 2), RELATION('d', 1), RELATION('d', 2),

RELATION('e', 1), RELATION('e', 2), RELATION('f', 1), RELATION('f', 2),

RELATION('g', 1), RELATION('g', 2), RELATION('h', 1), RELATION('h', 2),

RELATION('i', 1), RELATION('i', 2), RELATION('j', 1), RELATION('j', 2),

RELATION('k', 1), RELATION('k', 2), RELATION('l', 1), RELATION('l', 2),

RELATION('m', 1), RELATION('m', 2), RELATION('n', 1), RELATION('n', 2),

RELATION('o', 1), RELATION('o', 2), RELATION('p', 1), RELATION('p', 2),

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

RELATION('s', 1), RELATION('s', 2), RELATION('t', 1), RELATION('t', 2),

RELATION('u', 1), RELATION('u', 2), RELATION('v', 1), RELATION('v', 2),

RELATION('w', 1), RELATION('w', 2), RELATION('x', 1), RELATION('x', 2),

RELATION('y', 1), RELATION('y', 2), RELATION('z', 1), RELATION('z', 2),

RELATION('А', 1), RELATION('А', 2), RELATION('Б', 1), RELATION('Б', 2),

RELATION('В', 1), RELATION('В', 2), RELATION('Г', 1), RELATION('Г', 2),

RELATION('Д', 1), RELATION('Д', 2), RELATION('Е', 1), RELATION('Е', 2),

RELATION('Ё', 1), RELATION('Ё', 2), RELATION('Ж', 1), RELATION('Ж', 2),

RELATION('З', 1), RELATION('З', 2), RELATION('И', 1), RELATION('И', 2),

RELATION('Й', 1), RELATION('Й', 2), RELATION('К', 1), RELATION('К', 2),

RELATION('Л', 1), RELATION('Л', 2), RELATION('М', 1), RELATION('М', 2),

RELATION('Н', 1), RELATION('Н', 2), RELATION('О', 1), RELATION('О', 2),

RELATION('П', 1), RELATION('П', 2), RELATION('Р', 1), RELATION('Р', 2),

RELATION('С', 1), RELATION('С', 2), RELATION('Т', 1), RELATION('Т', 2),

RELATION('У', 1), RELATION('У', 2), RELATION('Ф', 1), RELATION('Ф', 2),

RELATION('Х', 1), RELATION('Х', 2), RELATION('Ц', 1), RELATION('Ц', 2),

RELATION('Ч', 1), RELATION('Ч', 2), RELATION('Ш', 1), RELATION('Ш', 2),

RELATION('Щ', 1), RELATION('Щ', 2), RELATION('Ъ', 1), RELATION('Ъ', 2),

RELATION('Ы', 1), RELATION('Ы', 2), RELATION('Ь', 1), RELATION('Ь', 2),

RELATION('Э', 1), RELATION('Э', 2), RELATION('Ю', 1), RELATION('Ю', 2),

RELATION('Я', 1), RELATION('Я', 2), RELATION('а', 1), RELATION('а', 2),

RELATION('б', 1), RELATION('б', 2), RELATION('в', 1), RELATION('в', 2),

RELATION('г', 1), RELATION('г', 2), RELATION('д', 1), RELATION('д', 2),

RELATION('е', 1), RELATION('е', 2), RELATION('ё', 1), RELATION('ё', 2),

RELATION('ж', 1), RELATION('ж', 2), RELATION('з', 1), RELATION('з', 2),

RELATION('и', 1), RELATION('и', 2), RELATION('й', 1), RELATION('й', 2),

RELATION('к', 1), RELATION('к', 2), RELATION('л', 1), RELATION('л', 2),

RELATION('м', 1), RELATION('м', 2), RELATION('н', 1), RELATION('н', 2),

RELATION('о', 1), RELATION('о', 2), RELATION('п', 1), RELATION('п', 2),

RELATION('р', 1), RELATION('р', 2), RELATION('с', 1), RELATION('с', 2),

RELATION('т', 1), RELATION('т', 2), RELATION('у', 1), RELATION('у', 2),

RELATION('ф', 1), RELATION('ф', 2), RELATION('х', 1), RELATION('х', 2),

RELATION('ц', 1), RELATION('ц', 2), RELATION('ч', 1), RELATION('ч', 2),

RELATION('ш', 1), RELATION('ш', 2), RELATION('щ', 1), RELATION('щ', 2),

RELATION('ъ', 1), RELATION('ъ', 2), RELATION('ы', 1), RELATION('ы', 2),

RELATION('ь', 1), RELATION('ь', 2), RELATION('э', 1), RELATION('э', 2),

RELATION('ю', 1), RELATION('ю', 2), RELATION('я', 1), RELATION('я', 2),

RELATION('0', 1), RELATION('0', 2), RELATION('1', 1), RELATION('1', 2),

RELATION('2', 1), RELATION('2', 2), RELATION('3', 1), RELATION('3', 2),

RELATION('4', 1), RELATION('4', 2), RELATION('5', 1), RELATION('5', 2),

RELATION('6', 1), RELATION('6', 2), RELATION('7', 1), RELATION('7', 2),

RELATION('8', 1), RELATION('8', 2), RELATION('9', 1), RELATION('9', 2),

RELATION(' ', 1), RELATION(' ', 2)),

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

NODE()

);

if (execute(strLit)) {

return 's';

}

FST intLit(

word,

7,

NODE(20,

RELATION('0', 1), RELATION('0', 6), // Если встречается "0", переход в состояние 1

RELATION('1', 5), RELATION('2', 5), RELATION('3', 5), RELATION('4', 5),

RELATION('5', 5), RELATION('6', 5), RELATION('7', 5), RELATION('8', 5), RELATION('9', 5),

RELATION('1', 6), RELATION('2', 6), RELATION('3', 6), RELATION('4', 6),

RELATION('5', 6), RELATION('6', 6), RELATION('7', 6), RELATION('8', 6), RELATION('9', 6)

),

// Состояние 1: После "0"

NODE(13,

RELATION('x', 3), RELATION('X', 3), // Если "x", то переходим к шестнадцатеричному формату

RELATION('b', 4), RELATION('B', 4), // Если "b", то переходим к двоичному формату

RELATION('0', 1), RELATION('0', 2), RELATION('1', 2), RELATION('2', 2), RELATION('3', 2), RELATION('4', 2),

RELATION('5', 2), RELATION('6', 2), RELATION('7', 2) // Восьмеричные цифры

),

NODE(16,

RELATION('0', 2), RELATION('1', 2), RELATION('2', 2), RELATION('3', 2), RELATION('4', 2),

RELATION('5', 2), RELATION('6', 2), RELATION('7', 2),

RELATION('0', 6), RELATION('1', 6), RELATION('2', 6), RELATION('3', 6), RELATION('4', 6),

RELATION('5', 6), RELATION('6', 6), RELATION('7', 6) // Восьмеричные цифры

),

// Состояние 2: После "0x"

NODE(44,

// Для "0x": допустимы только шестнадцатеричные символы

RELATION('0', 3), RELATION('1', 3), RELATION('2', 3), RELATION('3', 3), RELATION('4', 3), RELATION('5', 3),

RELATION('6', 3), RELATION('7', 3), RELATION('8', 3), RELATION('9', 3),

RELATION('a', 3), RELATION('b', 3), RELATION('c', 3), RELATION('d', 3),

RELATION('e', 3), RELATION('f', 3), RELATION('A', 3), RELATION('B', 3),

RELATION('C', 3), RELATION('D', 3), RELATION('E', 3), RELATION('F', 3),

RELATION('0', 6), RELATION('1', 6), RELATION('2', 6), RELATION('3', 6), RELATION('4', 6), RELATION('5', 6),

RELATION('6', 6), RELATION('7', 6), RELATION('8', 6), RELATION('9', 6),

RELATION('a', 6), RELATION('b', 6), RELATION('c', 6), RELATION('d', 6),

RELATION('e', 6), RELATION('f', 6), RELATION('A', 6), RELATION('B', 6),

RELATION('C', 6), RELATION('D', 6), RELATION('E', 6), RELATION('F', 6)

),

NODE(4,

RELATION('0', 4), RELATION('1', 4),

RELATION('0', 6), RELATION('1', 6)

),

NODE(20,

RELATION('0', 5), RELATION('1', 5), RELATION('2', 5), RELATION('3', 5), RELATION('4', 5),

RELATION('5', 5), RELATION('6', 5), RELATION('7', 5), RELATION('8', 5), RELATION('9', 5),

RELATION('0', 6), RELATION('1', 6), RELATION('2', 6), RELATION('3', 6), RELATION('4', 6),

RELATION('5', 6), RELATION('6', 6), RELATION('7', 6), RELATION('8', 6), RELATION('9', 6)

),

NODE()

);

if (execute(intLit)) {

return 'i';

}

FST iLex(

word,

3,

NODE(156,

RELATION('A', 0), RELATION('A', 1), RELATION('A', 2),

RELATION('B', 0), RELATION('B', 1), RELATION('B', 2),

RELATION('C', 0), RELATION('C', 1), RELATION('C', 2),

RELATION('D', 0), RELATION('D', 1), RELATION('D', 2),

RELATION('E', 0), RELATION('E', 1), RELATION('E', 2),

RELATION('F', 0), RELATION('F', 1), RELATION('F', 2),

RELATION('G', 0), RELATION('G', 1), RELATION('G', 2),

RELATION('H', 0), RELATION('H', 1), RELATION('H', 2),

RELATION('I', 0), RELATION('I', 1), RELATION('I', 2),

RELATION('J', 0), RELATION('J', 1), RELATION('J', 2),

RELATION('K', 0), RELATION('K', 1), RELATION('K', 2),

RELATION('L', 0), RELATION('L', 1), RELATION('L', 2),

RELATION('M', 0), RELATION('M', 1), RELATION('M', 2),

RELATION('N', 0), RELATION('N', 1), RELATION('N', 2),

RELATION('O', 0), RELATION('O', 1), RELATION('O', 2),

RELATION('P', 0), RELATION('P', 1), RELATION('P', 2),

RELATION('Q', 0), RELATION('Q', 1), RELATION('Q', 2),

RELATION('R', 0), RELATION('R', 1), RELATION('R', 2),

RELATION('S', 0), RELATION('S', 1), RELATION('S', 2),

RELATION('T', 0), RELATION('T', 1), RELATION('T', 2),

RELATION('U', 0), RELATION('U', 1), RELATION('U', 2),

RELATION('V', 0), RELATION('V', 1), RELATION('V', 2),

RELATION('W', 0), RELATION('W', 1), RELATION('W', 2),

RELATION('X', 0), RELATION('X', 1), RELATION('X', 2),

RELATION('Y', 0), RELATION('Y', 1), RELATION('Y', 2),

RELATION('Z', 0), RELATION('Z', 1), RELATION('Z', 2),

RELATION('a', 0), RELATION('a', 1), RELATION('a', 2),

RELATION('b', 0), RELATION('b', 1), RELATION('b', 2),

RELATION('c', 0), RELATION('c', 1), RELATION('c', 2),

RELATION('d', 0), RELATION('d', 1), RELATION('d', 2),

RELATION('e', 0), RELATION('e', 1), RELATION('e', 2),

RELATION('f', 0), RELATION('f', 1), RELATION('f', 2),

RELATION('g', 0), RELATION('g', 1), RELATION('g', 2),

RELATION('h', 0), RELATION('h', 1), RELATION('h', 2),

RELATION('i', 0), RELATION('i', 1), RELATION('i', 2),

RELATION('j', 0), RELATION('j', 1), RELATION('j', 2),

RELATION('k', 0), RELATION('k', 1), RELATION('k', 2),

RELATION('l', 0), RELATION('l', 1), RELATION('l', 2),

RELATION('m', 0), RELATION('m', 1), RELATION('m', 2),

RELATION('n', 0), RELATION('n', 1), RELATION('n', 2),

RELATION('o', 0), RELATION('o', 1), RELATION('o', 2),

RELATION('p', 0), RELATION('p', 1), RELATION('p', 2),

RELATION('q', 0), RELATION('q', 1), RELATION('q', 2),

RELATION('r', 0), RELATION('r', 1), RELATION('r', 2),

RELATION('s', 0), RELATION('s', 1), RELATION('s', 2),

RELATION('t', 0), RELATION('t', 1), RELATION('t', 2),

RELATION('u', 0), RELATION('u', 1), RELATION('u', 2),

RELATION('v', 0), RELATION('v', 1), RELATION('v', 2),

RELATION('w', 0), RELATION('w', 1), RELATION('w', 2),

RELATION('x', 0), RELATION('x', 1), RELATION('x', 2),

RELATION('y', 0), RELATION('y', 1), RELATION('y', 2),

RELATION('z', 0), RELATION('z', 1), RELATION('z', 2),

RELATION('\_', 0), RELATION('\_', 1)

),

NODE(126,

RELATION('0', 1), RELATION('0', 2),

RELATION('1', 1), RELATION('1', 2),

RELATION('2', 1), RELATION('2', 2),

RELATION('3', 1), RELATION('3', 2),

RELATION('4', 1), RELATION('4', 2),

RELATION('5', 1), RELATION('5', 2),

RELATION('6', 1), RELATION('6', 2),

RELATION('7', 1), RELATION('7', 2),

RELATION('8', 1), RELATION('8', 2),

RELATION('9', 1), RELATION('9', 2),

RELATION('\_', 1), RELATION('\_', 2),

RELATION('A', 1), RELATION('A', 2),

RELATION('B', 1), RELATION('B', 2),

RELATION('C', 1), RELATION('C', 2),

RELATION('D', 1), RELATION('D', 2),

RELATION('E', 1), RELATION('E', 2),

RELATION('F', 1), RELATION('F', 2),

RELATION('G', 1), RELATION('G', 2),

RELATION('H', 1), RELATION('H', 2),

RELATION('I', 1), RELATION('I', 2),

RELATION('J', 1), RELATION('J', 2),

RELATION('K', 1), RELATION('K', 2),

RELATION('L', 1), RELATION('L', 2),

RELATION('M', 1), RELATION('M', 2),

RELATION('N', 1), RELATION('N', 2),

RELATION('O', 1), RELATION('O', 2),

RELATION('P', 1), RELATION('P', 2),

RELATION('Q', 1), RELATION('Q', 2),

RELATION('R', 1), RELATION('R', 2),

RELATION('S', 1), RELATION('S', 2),

RELATION('T', 1), RELATION('T', 2),

RELATION('U', 1), RELATION('U', 2),

RELATION('V', 1), RELATION('V', 2),

RELATION('W', 1), RELATION('W', 2),

RELATION('X', 1), RELATION('X', 2),

RELATION('Y', 1), RELATION('Y', 2),

RELATION('Z', 1), RELATION('Z', 2),

RELATION('a', 1), RELATION('a', 2),

RELATION('b', 1), RELATION('b', 2),

RELATION('c', 1), RELATION('c', 2),

RELATION('d', 1), RELATION('d', 2),

RELATION('e', 1), RELATION('e', 2),

RELATION('f', 1), RELATION('f', 2),

RELATION('g', 1), RELATION('g', 2),

RELATION('h', 1), RELATION('h', 2),

RELATION('i', 1), RELATION('i', 2),

RELATION('j', 1), RELATION('j', 2),

RELATION('k', 1), RELATION('k', 2),

RELATION('l', 1), RELATION('l', 2),

RELATION('m', 1), RELATION('m', 2),

RELATION('n', 1), RELATION('n', 2),

RELATION('o', 1), RELATION('o', 2),

RELATION('p', 1), RELATION('p', 2),

RELATION('q', 1), RELATION('q', 2),

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

RELATION('s', 1), RELATION('s', 2),

RELATION('t', 1), RELATION('t', 2),

RELATION('u', 1), RELATION('u', 2),

RELATION('v', 1), RELATION('v', 2),

RELATION('w', 1), RELATION('w', 2),

RELATION('x', 1), RELATION('x', 2),

RELATION('y', 1), RELATION('y', 2),

RELATION('z', 1), RELATION('z', 2)),

NODE()

);

Листинг Б.3 – Конечные автоматы

# ПРИЛОЖЕНИЕ В

Greibach greiibach(NS('S'), TS('$'),

6,

Rule(NS('S'), GRB\_ERROR\_SERIES + 0,

12,

Rule::Chain(8, TS('m'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), NS('S')),

Rule::Chain(7, TS('m'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}')),

Rule::Chain(7, TS('m'), TS('{'), TS('r'), NS('E'), TS(';'), TS('}'), NS('S')),

Rule::Chain(6, TS('m'), TS('{'), TS('r'), NS('E'), TS(';'), TS('}')),

Rule::Chain(13, TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), NS('S')),

Rule::Chain(12, TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}')),

Rule::Chain(12, TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), TS('r'), NS('E'), TS(';'), TS('}'), NS('S')),

Rule::Chain(11, TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), TS('r'), NS('E'), TS(';'), TS('}')),

Rule::Chain(12, TS('t'), TS('f'), TS('i'), TS('('), TS(')'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}'), NS('S')),

Rule::Chain(11, TS('t'), TS('f'), TS('i'), TS('('), TS(')'), TS('{'), NS('N'), TS('r'), NS('E'), TS(';'), TS('}')),

Rule::Chain(11, TS('t'), TS('f'), TS('i'), TS('('), TS(')'), TS('{'), TS('r'), NS('E'), TS(';'), TS('}'), NS('S')),

Rule::Chain(10, TS('t'), TS('f'), TS('i'), TS('('), TS(')'), TS('{'), TS('r'), NS('E'), TS(';'), TS('}'))

),

Rule(NS('N'), GRB\_ERROR\_SERIES + 1,

14,

Rule::Chain(4, TS('d'), TS('t'), TS('i'), TS(';')),

Rule::Chain(7, TS('d'), TS('t'), TS('i'), TS('='), NS('E'), TS(';')),

Rule::Chain(4, TS('i'), TS('='), NS('E'), TS(';')),

Rule::Chain(3, TS('r'), NS('E'), TS(';')),

Rule::Chain(5, TS('d'), TS('t'), TS('i'), TS(';'), NS('N')),

Rule::Chain(7, TS('d'), TS('t'), TS('i'), TS('='), NS('E'), TS(';'), NS('N')),

Rule::Chain(4, TS('r'), NS('E'), TS(';'), NS('N')),

Rule::Chain(5, TS('i'), TS('='), NS('E'), TS(';'), NS('N')),

Rule::Chain(3, TS('p'), NS('E'), TS(';')),

Rule::Chain(4, TS('p'), NS('E'), TS(';'), NS('N')),

Rule::Chain(3, TS('n'), NS('i'), TS(';')),

Rule::Chain(4, TS('n'), NS('i'), TS(';'), NS('N')),

Rule::Chain(7, TS('?'), TS('('), NS('E'), TS(')'), TS('{'), NS('N'), TS('}')),

Rule::Chain(8, TS('?'), TS('('), NS('E'), TS(')'), TS('{'), NS('N'), TS('}'), NS('N'))

),

Rule(NS('E'), GRB\_ERROR\_SERIES + 2,

9,

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(NS('W'), GRB\_ERROR\_SERIES + 3,

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('F'), GRB\_ERROR\_SERIES + 4,

2,

Rule::Chain(2, TS('t'), TS('i')),

Rule::Chain(4, TS('t'), TS('i'), TS(','), NS('F'))

),

Rule(NS('M'), GRB\_ERROR\_SERIES + 2,

8,

Rule::Chain(2, TS('|'), NS('E')),

Rule::Chain(3, TS('|'), NS('E'), NS('M')),

Rule::Chain(2, TS('&'), NS('E')),

Rule::Chain(3, TS('&'), NS('E'), NS('M')),

Rule::Chain(2, TS('!'), NS('E')),

Rule::Chain(3, TS('!'), NS('E'), NS('M')),

Rule::Chain(2, TS('^'), NS('E')),

Rule::Chain(3, TS('^'), NS('E'), NS('M'))

)

);

Листинг В.1 – Правила, описывающие грамматику языка

# ПРИЛОЖЕНИЕ Г

struct Mfst

{

enum RC\_STEP {

NS\_OK,

NS\_NORULE,

NS\_NORULECHAIN,

NS\_ERROR,

TS\_OK,

TS\_NOK,

LENTA\_END,

SURPRISE,

};

struct MfstDiagnosis

{

short lenta\_position;

RC\_STEP rc\_step;

short nrule;

short nrule\_chain;

MfstDiagnosis();

MfstDiagnosis(short plenta\_position, RC\_STEP prc\_step, short pnrule, short pnrule\_chain);

} diagnosis[MFST\_DIAGN\_NUMBER];

GRBALPHABET\* lenta;

short lenta\_position;

short nrule;

short nrulechain;

short lenta\_size;

GRB::Greibach grebach;

LT::LexTable lex;

MFSTSTSTACK st;

my\_stack\_MfstState storestate;

Mfst();

Mfst(LT::LexTable& plex, GRB::Greibach pgrebach);

char\* getCSt(char\* buf);

char\* getCLenta(char\* buf, short pos, short n = 25);

char\* getDiagnosis(short n, char\* buf, Log::LOG log);

bool savestate();

bool resetstate();

bool push\_chain(GRB::Rule::Chain chain);

RC\_STEP step();

bool start(Log::LOG log);

bool savediagnosis(RC\_STEP pprc\_step);

void printrules(Log::LOG log);

struct Deducation

{

short size;

short\* nrules;

short\* nrulechains;

Deducation()

{

size = 0;

nrules = 0;

nrulechains = 0;

};

} deducation;

bool savededucation();

};

Листинг Г.1 – Структура магазинного конечного автомата

# ПРИЛОЖЕНИЕ Д

0 : S->tfi(F){NrE;}S tfi(ti,ti){dti;i=i&(i|i); S$

0 : SAVESTATE: 1

0 : tfi(ti,ti){dti;i=i&(i|i); tfi(F){NrE;}S$

1 : fi(ti,ti){dti;i=i&(i|i);r fi(F){NrE;}S$

2 : i(ti,ti){dti;i=i&(i|i);ri i(F){NrE;}S$

3 : (ti,ti){dti;i=i&(i|i);ri; (F){NrE;}S$

4 : ti,ti){dti;i=i&(i|i);ri;} F){NrE;}S$

5 : F->ti ti,ti){dti;i=i&(i|i);ri;} F){NrE;}S$

5 : SAVESTATE: 2

5 : ti,ti){dti;i=i&(i|i);ri;} ti){NrE;}S$

6 : i,ti){dti;i=i&(i|i);ri;}m i){NrE;}S$

7 : ,ti){dti;i=i&(i|i);ri;}m{ ){NrE;}S$

8 : TS\_NOK / NS\_NORULECHAIN

8 : RESSTATE

Листинг Д.1 – Работа синтаксического анализатора (начало разбора)

TS\_NOK / NS\_NORULECHAIN

399 : RESSTATE

399 : l;ni;?(l){i=!i;pi;}rl;} E;rE;}S$

400 : E->lM l;ni;?(l){i=!i;pi;}rl;} E;rE;}S$

400 : SAVESTATE: 42

400 : l;ni;?(l){i=!i;pi;}rl;} lM;rE;}S$

401 : ;ni;?(l){i=!i;pi;}rl;} M;rE;}S$

402 : NS\_NORULECHAIN/NS\_NORULE

402 : RESSTATE

402 : l;ni;?(l){i=!i;pi;}rl;} E;rE;}S$

403 : NS\_NORULECHAIN/NS\_NORULE

403 : RESSTATE

403 : pl;ni;?(l){i=!i;pi;}rl;} NrE;}S$

404 : N->pE;N pl;ni;?(l){i=!i;pi;}rl;} NrE;}S$

404 : SAVESTATE: 41

404 : pl;ni;?(l){i=!i;pi;}rl;} pE;NrE;}S$

405 : l;ni;?(l){i=!i;pi;}rl;} E;NrE;}S$

406 : E->l l;ni;?(l){i=!i;pi;}rl;} E;NrE;}S$

406 : SAVESTATE: 42

406 : l;ni;?(l){i=!i;pi;}rl;} l;NrE;}S$

407 : ;ni;?(l){i=!i;pi;}rl;} ;NrE;}S$

408 : ni;?(l){i=!i;pi;}rl;} NrE;}S$

409 : N->ni; ni;?(l){i=!i;pi;}rl;} NrE;}S$

409 : SAVESTATE: 43

409 : ni;?(l){i=!i;pi;}rl;} ni;rE;}S$

410 : i;?(l){i=!i;pi;}rl;} i;rE;}S$

411 : ------>NS\_ERROR

Листинг Д.2 – Работа синтаксического анализатора (конец работы)

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

4 : F->ti,F

7 : F->ti

11 : N->dti;N

15 : N->i=E;

17 : E->iM

18 : M->&E

19 : E->(E)

20 : E->iM

21 : M->|E

22 : E->i

26 : E->i

29 : S->m{NrE;}S

31 : N->dti;N

35 : N->dti=E;N

39 : E->l

41 : N->dti;N

45 : N->dti;N

49 : N->i=E;N

51 : E->l

53 : N->i=E;N

55 : E->l

57 : N->i=E;N

59 : E->l

61 : N->i=E;N

63 : E->i(W)

65 : W->i,W

67 : W->i

70 : N->pE;N

71 : E->l

73 : N->ni;

101 : N->ni;

Листинг Д.3 – Результат работы синтаксического анализатора

# ПРИЛОЖЕНИЕ Д

bool PolishNotation(int lextable\_pos, LT::LexTable& lextable, IT::IdTable& idtable) {

std::map<char, int> precedence = { {'!', 1}, {'&', 2}, {'|', 3}, {'^', 2} };

std::set<char> left\_associative = { '!', '&', '|', '^' };

LT::LexTable expression = LT::Create(LEN);

int length = 0;

for (int i = lextable\_pos; lextable.table[i].lexema != ';'; i++) {

LT::Add(expression, lextable.table[i]);

length++;

}

std::stack<LT::Entry> stack;

LT::LexTable output = LT::Create(LEN);

for (int i = 0; i <= length; i++) {

if (std::isalnum(expression.table[i].lexema)) {

if (idtable.table[expression.table[i].idxTI].idtype == IT::F) {

stack.push(expression.table[i]);

}

else {

LT::Add(output, expression.table[i]);

}

}

else if (expression.table[i].lexema == '(') {

stack.push(expression.table[i]);

}

else if (expression.table[i].lexema == ')') {

while (!stack.empty() && stack.top().lexema != '(') {

LT::Add(output, stack.top());

stack.pop();

}

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

stack.pop();

}

}

else if (precedence.find(expression.table[i].lexema) != precedence.end()) {

LT::Entry op = expression.table[i];

while (!stack.empty() && stack.top().lexema != '(' &&

(precedence[stack.top().lexema] > precedence[op.lexema] ||

(precedence[stack.top().lexema] == precedence[op.lexema] && left\_associative.count(op.lexema)))) {

LT::Add(output, stack.top());

stack.pop();

}

stack.push(op);

}

}

while (!stack.empty()) {

LT::Add(output, stack.top());

stack.pop();

}

for (int i = output.size; i < length; i++) {

LT::Add(output, { '$', LT::GetEntry(output, output.size - 1).sn });

}

for (int i = 0; i < length; i++) {

lextable.table[i + lextable\_pos] = output.table[i];

}

return true;

}

Листинг Е.1 – Алгоритм преобразования выражений к польской записи

# ПРИЛОЖЕНИЕ Ж

;========================= шапка ==============================

.586

.model flat, stdcall

includelib libucrt.lib

includelib kernel32.lib

includelib standartLib.lib

ExitProcess PROTO : DWORD

EXTRN rusL: proc

EXTRN OutputInt: proc

EXTRN OutputChar: proc

EXTRN InputInt: proc

EXTRN strEq: proc

EXTRN findWord: proc

.stack 4096

;=================== сегмент констант =========================

.const

INT00GLOBAL DWORD 45

BOOl10GLOBAL BYTE 0

INT20GLOBAL DWORD 39

STR30GLOBAL BYTE '1234567890', 0

STR40GLOBAL BYTE 'контрольный пример', 0

BOOl50GLOBAL BYTE 1

INT60GLOBAL DWORD 0

;=================== сегмент данных ============================

.data

buffer BYTE 256 dup(0)

zfi0GLOBAL DWORD ?

xMAIN DWORD ?

zMAIN DWORD ?

saMAIN DWORD ?

bMAIN DWORD ?

;=================== сегмент кода ==============================

.code

fi0GLOBAL PROC xfi0GLOBAL : DWORD, yfi0GLOBAL : DWORD

push xfi0GLOBAL

push xfi0GLOBAL

push yfi0GLOBAL

pop eax

pop ebx

or eax, ebx

push eax

pop eax

pop ebx

and eax, ebx

push eax

pop zfi0GLOBAL

push strEq

jmp local0

local0:

pop eax

ret

fi0GLOBAL ENDP

main PROC

call rusL

push INT00GLOBAL

pop zMAIN

mov al, BOOl10GLOBAL

push ax

pop bMAIN

push INT20GLOBAL

pop xMAIN

push offset STR30GLOBAL

pop saMAIN

push xMAIN

push zMAIN

call fi0GLOBAL

push eax

push fi0GLOBAL

pop zMAIN

push ecx

push offset STR40GLOBAL

call OutputChar

pop ecx

pop ecx

push ecx

call InputInt

mov zMAIN, eax

pop ecx

mov al, BOOl50GLOBAL

cmp al, 1

jz m0

jnz m1

je m0

m0:

push zMAIN

pop eax

not eax

push eax

pop zMAIN

push ecx

push zMAIN

call OutputInt

pop ecx

pop ecx

m1:

push strEq

jmp theend

theend:

call ExitProcess

main ENDP

end main

;=============================================================

Листинг Ж.1 – Результат генерации ассемблерного кода

# ПРИЛОЖЕНИЕ З

Тест без ошибок

Тест без ошибок

--—- Протокол ——--

Дата: 19.12.24 03:41:54

--—- Параметры ——--

-in: in.kvv

-out: in.kvv.asm

-log: in.kvv.log

--—- Исходные данные ——--

Кол-во символов: 305

Кол-во строк: 25

Пропущенно: 17

------Преобразованный код-------

integer function fi(integer x, integer y)

{

declare integer z;

z= x & (x|y);

return z;

}

main

{

declare integer x;

declare integer z = 45;

declare char sa;

declare bool b;

b = false;

x = 047;

sa = '1234567890';

z = fi(x,z);

write 'контрольный пример';

read z;

if (true){

z = !z;

write z;

}

return 0;

}

--—- Представления кода в виде лексем ——--

01 tfi|7|(ti|8|,ti|9|)

02 {

03 dti|10|;

04 i|10|=i|8|&(i|8||i|9|);

05 ri|10|;

06 }

07 m

08 {

09 dti|11|;

10 dti|12|=l;

11 dti|14|;

12 dti|15|;

13 i|15|=l;

14 i|11|=l;

15 i|14|=l;

16 i|12|=i|7|(i|11|,i|12|);

17 pl;

18 ni|12|;

19 ?(l){

20 i|12|=!i|12|;

21 pi|12|;

22 }

23 rl;

24 }

--—------------------------------------------- Идентификаторы ---------------------------------------——-----------------

| Номер | id | Тип данных | Тип | Связь (Номер Лексема Строка) | Видимость | Значение ?

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

| 1 | strEq | bool | функция | 0 $ 0 | 0GLOBAL |

| 2 | a | str | параметр | 0 $ 0 | strEq -> 0GLOBAL |

| 3 | b | str | параметр | 0 $ 0 | strEq -> 0GLOBAL |

| 4 | findWord | str | функция | 0 $ 0 | 0GLOBAL |

| 5 | a | str | параметр | 0 $ 0 | findWord -> 0GLOBAL |

| 6 | n | int | параметр | 0 $ 0 | findWord -> 0GLOBAL |

| 7 | fi | int | функция | 3 i 1 | 0GLOBAL |

| 8 | x | int | параметр | 6 i 1 | fi -> 0GLOBAL |

| 9 | y | int | параметр | 9 i 1 | fi -> 0GLOBAL |

| 10 | z | int | переменная | 14 i 3 | fi -> 0GLOBAL |

| 11 | x | int | переменная | 34 i 9 | MAIN -> 0GLOBAL |

| 12 | z | int | переменная | 38 i 10 | MAIN -> 0GLOBAL |

| 13 | INT0 | int | литерал | 40 l 10 | 0GLOBAL | 45

| 14 | sa | str | переменная | 44 i 11 | MAIN -> 0GLOBAL |

| 15 | b | bool | переменная | 48 i 12 | MAIN -> 0GLOBAL |

| 16 | BOOl1 | bool | литерал | 52 l 13 | 0GLOBAL | 0

| 17 | INT2 | int | литерал | 56 l 14 | 0GLOBAL | 39

| 18 | STR3 | str | литерал | 60 l 15 | 0GLOBAL | 10 '1234567890'

| 19 | STR4 | str | литерал | 72 l 17 | 0GLOBAL | 20 'контрольный пример'

| 20 | BOOl5 | bool | литерал | 79 l 19 | 0GLOBAL | 1

| 21 | INT6 | int | литерал | 92 l 23 | 0GLOBAL | 0

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

----------Правила-----------

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

4 : F->ti,F

7 : F->ti

11 : N->dti;N

15 : N->i=E;

17 : E->iM

18 : M->&E

19 : E->(E)

20 : E->iM

21 : M->|E

22 : E->i

26 : E->i

29 : S->m{NrE;}S

31 : N->dti;N

35 : N->dti=E;N

39 : E->l

41 : N->dti;N

45 : N->dti;N

49 : N->i=E;N

51 : E->l

53 : N->i=E;N

55 : E->l

57 : N->i=E;N

59 : E->l

61 : N->i=E;N

63 : E->i(W)

65 : W->i,W

67 : W->i

70 : N->pE;N

71 : E->l

73 : N->ni;

Сем анализ пройден

польская запись построена

--—- Представления кода в виде лексем ——--

10 tfi|7|(ti|8|,ti|9|)

20 {

30 dti|10|;

40 i|10|=i|8|i|8|i|9||&$$;

50 ri|10|;

60 }

70 m

80 {

90 dti|11|;

10 dti|12|=l;

11 dti|14|;

12 dti|15|;

13 i|15|=l;

14 i|11|=l;

15 i|14|=l;

16 i|12|=i|11|i|12|i|7|$$$;

17 pl;

18 ni|12|;

19 ?(l){

20 i|12|=i|12|!;

21 pi|12|;

22 }

23 rl;

24 }

--—------------------------------------------- Идентификаторы ---------------------------------------——-----------------

| Номер | id | Тип данных | Тип | Связь (Номер Лексема Строка) | Видимость | Значение ?

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

| 1 | strEq | bool | функция | 0 $ 0 | 0GLOBAL |

| 2 | a | str | параметр | 0 $ 0 | strEq -> 0GLOBAL |

| 3 | b | str | параметр | 0 $ 0 | strEq -> 0GLOBAL |

| 4 | findWord | str | функция | 0 $ 0 | 0GLOBAL |

| 5 | a | str | параметр | 0 $ 0 | findWord -> 0GLOBAL |

| 6 | n | int | параметр | 0 $ 0 | findWord -> 0GLOBAL |

| 7 | fi | int | функция | 3 i 1 | 0GLOBAL |

| 8 | x | int | параметр | 6 i 1 | fi -> 0GLOBAL |

| 9 | y | int | параметр | 9 i 1 | fi -> 0GLOBAL |

| 10 | z | int | переменная | 14 i 3 | fi -> 0GLOBAL |

| 11 | x | int | переменная | 34 i 9 | MAIN -> 0GLOBAL |

| 12 | z | int | переменная | 38 i 10 | MAIN -> 0GLOBAL |

| 13 | INT0 | int | литерал | 40 l 10 | 0GLOBAL | 45

| 14 | sa | str | переменная | 44 i 11 | MAIN -> 0GLOBAL |

| 15 | b | bool | переменная | 48 i 12 | MAIN -> 0GLOBAL |

| 16 | BOOl1 | bool | литерал | 52 l 13 | 0GLOBAL | 0

| 17 | INT2 | int | литерал | 56 l 14 | 0GLOBAL | 39

| 18 | STR3 | str | литерал | 60 l 15 | 0GLOBAL | 10 '1234567890'

| 19 | STR4 | str | литерал | 72 l 17 | 0GLOBAL | 20 'контрольный пример'

| 20 | BOOl5 | bool | литерал | 79 l 19 | 0GLOBAL | 1

| 21 | INT6 | int | литерал | 92 l 23 | 0GLOBAL | 0

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

Листинг 8.1 – Вывод протокола