зхМИНИСТЕРСТВО 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

[1. Спецификация языка программирования 3](#_Toc181789081)

[1.1. Характеристика языка программирования 3](#_Toc181789082)

**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

|  |
| --- |
| <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 |

Таблица 1.4 – Идентификаторы KVV-2024

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.

|  |  |
| --- | --- |
| 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 |

Таблица 1.5. – Литералы KVV-2024

*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 | Print <идентификатор> | <литерал>; | print 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 |
| & | 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 1.7 - 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

1. Сравнение строк:
   * Назначение: сравнивает две строки и возвращает результат их сравнения (меньше, равно, больше).
   * Параметры: две строки (char\*).
   * Возвращаемое значение: целое число (int), где -1 означает "меньше", 0 — "равны", 1 — "больше".
2. Нахождение слова:
   * Назначение: находит слово в строке.
   * Параметры: строка и число.
   * Возвращаемое значение: строка.

Способ подключения и применение:

* Стандартная библиотека подключается автоматически компилятором.
* Вызов функций осуществляется через их имя с указанием необходимых параметров.

## 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 print. 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

integer function fi(integer x, integer y)

{

declare integer z;//я люлю коменты

z= x & (x|y);

return z;

}

main

{

declare integer x;

declare bool y;

declare integer z = 45;

declare char sa;

declare char sb;

declare bool b;

b = true;

b = false;

x = 0x12Ab4;

x = 0b01;

x = 047;

y = true;

sa = '1234567890';

sb = '1234567890';

z = fi(x,z);

write 'контрольный пример';

if (true){

read z;

}

write z;

return 0;

}

## 2. 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.

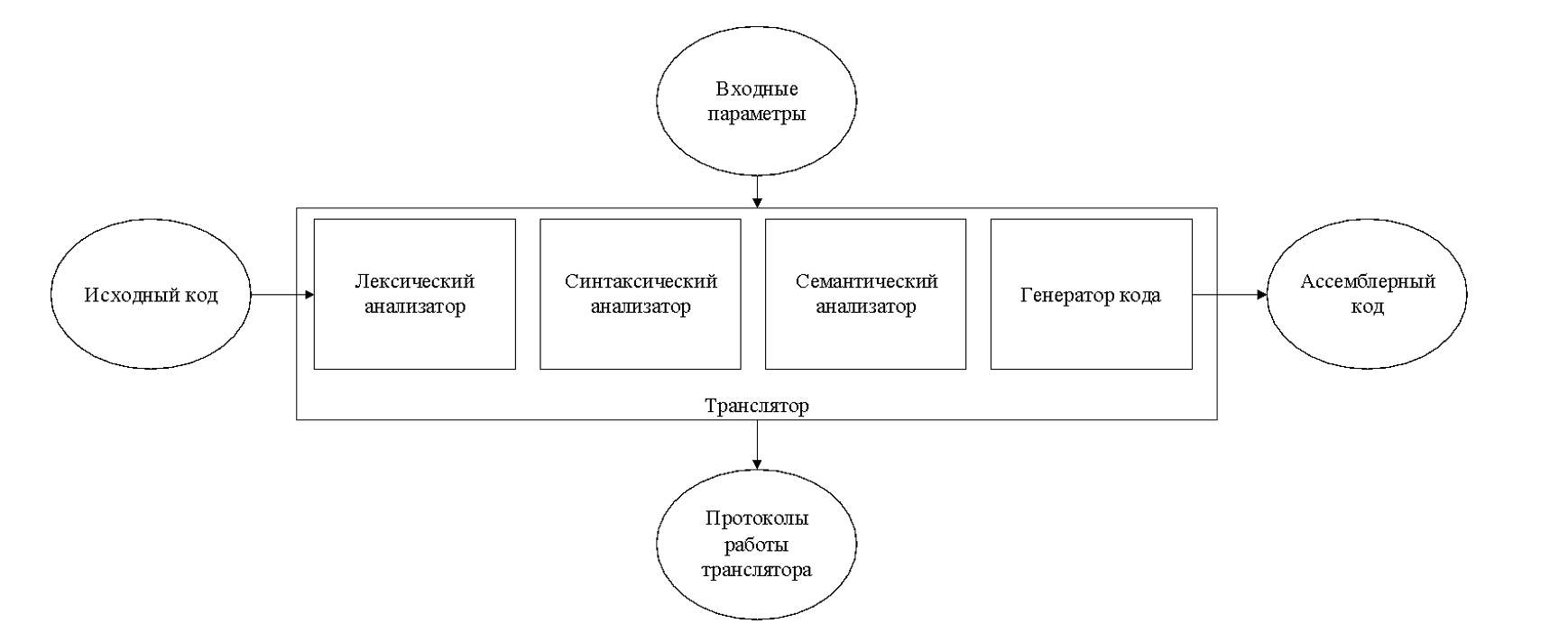


Рисунок 2.1 – Графическое отображение структуры транслятора

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.

## 3. 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.

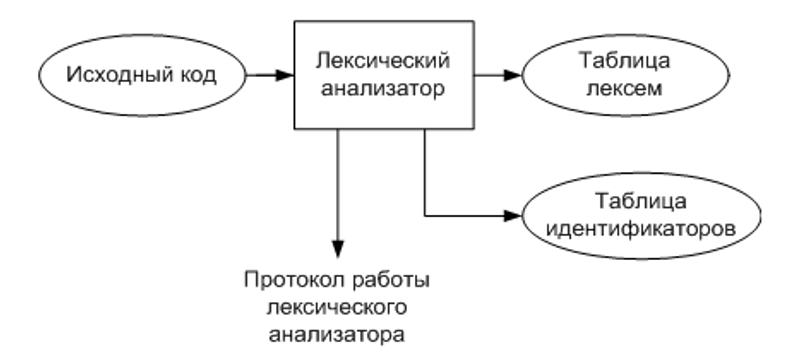


Рисунок 3.1 – Структура лексического анализатора

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.

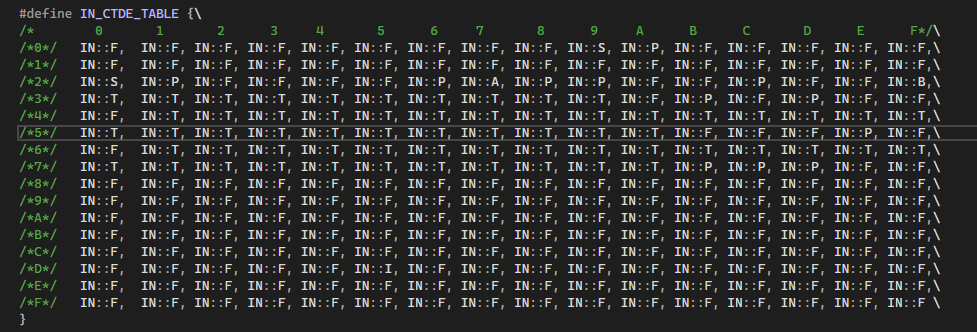


Рисунок 3.2 – Таблица допустимости входных символов

Эта таблица была сформирована на основе кодировки Windows-1251 и необходима для проверки допустимости символов. В таблице 3.3 приводятся символы и их значения.

Таблица 3.3 – 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:

1. Получение исходного текста программы;
2. Последовательное сканирование текста программы;
3. Проверка символов: при нахождении лишних пробелов или знака табуляции переходим к следующему символу;
4. Формирование результирующего текста, состоящего только из значимых символов;
5. Вывод результирующего текста, готового для дальнейшей обработки.

## 3.4 Перечень ключевых слов

Перечень ключевых слов и лексем приведены в таблице 3.4.

Таблица 3.4 – 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 | print | p |
| Блок функции | { | { |
| } | } |
| Оператор | Oперации | | | | |
| & | & |
| ! | ! |
| ^ | ^ |
| Оператор присваивания | = | = |
| Идентификатор | - | [a-z;A-Z]+  [a-z;A-Z;0-9]\* | i |
| Точка входа | - | main | m |
| Литерал | Строковый | [1-9]+[0-9]\* | l |
| Целочисленный | [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

Приложение Б включает ключевые структуры данных, используемые на этапе лексического анализа.

Таблица лексем содержит:

* Лексемы
* Номер строки, в которой находится каждая лексема
* Таблица идентификаторов содержит:
* Сам идентификатор
* Тип данных
* Тип идентификатора
* Индекс в таблице лексем
* Значение

## 3.6 Структура и перечень сообщений 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 3.5.

Таблица 3.5 – 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. |
| 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:

1. Установление начальных параметров, таких как текущая позиция в исходном коде;
2. Постепенное считывание символов из входного потока;
3. Для каждого прочитанного символа определение его типа (буква, цифра, пробел и т.д.);
4. Сборка символов в лексемы (токены) в зависимости от их типа и правил грамматики;
5. Определение типа каждой лексемы (ключевое слово, идентификатор, число и т.д.);
6. Если встречается некорректный символ или последовательность символов, формирование сообщения об ошибке;
7. Возвращение готовых лексем синтаксическому анализатору;
8. Когда весь исходный код обработан, передача специальных лексем, указывающих на конец входного потока.

## 3.10 Контрольный пример

## 4. Разработка синтаксического анализатора

## 4.1 Структура синтаксического анализатора

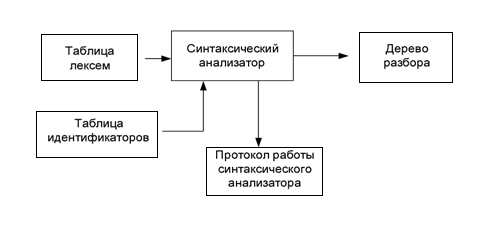
Синтаксический анализ — это этап трансляции, следующий за лексическим анализом и предназначенный для распознавания синтаксических конструкций. Структура синтаксического анализатора представлена на рисунке 4.1.

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

## 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) |
|  | 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.

## 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

## 4.7 Параметры синтаксического анализатора и режимы его работы

## 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

## 5. 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.

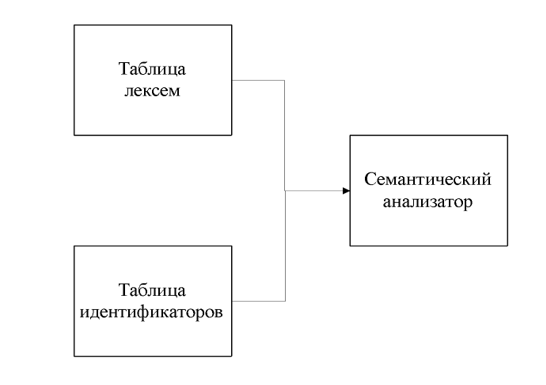


Рисунок 5.1 – Структура семантического анализатора

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

## 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

## 6. 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 6.2.

Таблица 6.2 – Пример преобразования выражение в обратную польскую запись

|  |  |  |
| --- | --- | --- |
| Исходная строка | Результирующая строка | Стек |
| x+7\*y/(z-4) |  |  |
| +7\*y/(z-4) | x |  |
| 7\*y/(z-4) | x | + |
| \*y/(z-4) | x7 | + |
| y/(z-4) | x7 | +\* |
| /(z-4) | x7y | +\* |
| (z-4) | x7y\* | +/ |
| z-4) | x7y\* | +/( |
| -4) | x7y\*z | +/( |
| 4) | x7y\*z | +/(- |
| ) | x7y\*z4 | +/(- |
|  | x7y\*z4- | +/ |
|  | x7y\*z4-/ | + |
|  | x7y\*z4-/+ |  |

## 6.3 Программная реализация обработки выражений

## 6.4 Контрольный пример

## 7. 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

## 7.3 Статическая библиотека

## 7.4 Особенности алгоритма генерации кода

## 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

## 8. Тестирование транслятора

## 8.1 Общие положения

## 8.2 Результаты тестирования

**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.