МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора KEM-2023»

Выполнил студент Козека Елизавета Максимовна

(Ф.И.О.)

Руководитель проекта ст. преп. Наркевич Аделина Сергеевна

(учен. степень, звание, должность, подпись, Ф.И.О.)

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

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты ст. преп. Наркевич Аделина Сергеевна

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер ст. преп. Наркевич Аделина Сергеевна

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2023

**Содержание**

[Введение 4](#_Toc122318388)

[Глава 1 Спецификация языка программирования 6](#_Toc122318389)

[1.1 Характеристика языка программирования 6](#_Toc122318390)

[1.2 Определение алфавита языка программирования 6](#_Toc122318391)

[1.3 Применяемые сепараторы 7](#_Toc122318392)

[1.4 Применяемые кодировки 7](#_Toc122318393)

[1.5 Типы данных 8](#_Toc122318394)

[1.6 Преобразование типов данных 9](#_Toc122318395)

[1.7 Идентификаторы 9](#_Toc122318396)

[1.8 Литералы 9](#_Toc122318397)

[1.9 Объявления данных 10](#_Toc122318398)

[1.10 Инициализация данных 11](#_Toc122318399)

[1.11 Инструкции языка 11](#_Toc122318400)

[1.12 Операции языка 12](#_Toc122318401)

[1.13 Выражения и их вычисления 12](#_Toc122318402)

[1.14 Конструкции языка 12](#_Toc122318403)

[1.15 Область видимости идентификаторов 13](#_Toc122318404)

[1.16 Семантические проверки 13](#_Toc122318405)

[1.17 Распределение оперативной памяти на этапе выполнения 14](#_Toc122318406)

[1.18 Стандартная библиотека и её состав 14](#_Toc122318407)

[1.19 Ввод и вывод данных 14](#_Toc122318408)

[1.20 Точка входа 15](#_Toc122318409)

[1.21 Препроцессор 15](#_Toc122318410)

[1.22 Соглашения о вызовах 15](#_Toc122318411)

[1.23 Объектный код 15](#_Toc122318412)

[1.24 Классификация сообщений транслятора 15](#_Toc122318413)

[1.25 Контрольный пример 15](#_Toc122318414)

[Глава 2 Структура транслятора 16](#_Toc122318415)

[2.1 Компоненты транслятора и их назначение 16](#_Toc122318416)

[2.2 Перечень входных параметров транслятора 17](#_Toc122318417)

[2.3 Протоколы, формируемые транслятором 17](#_Toc122318418)

[Глава 3 Разработка лексического анализатора 19](#_Toc122318419)

[3.1 Структура лексического анализатора 19](#_Toc122318420)

[3.2 Контроль входных символов 19](#_Toc122318421)

[3.3 Удаление избыточных символов 20](#_Toc122318422)

[3.4 Перечень ключевых слов 21](#_Toc122318423)

[3.5 Основные структуры данных 22](#_Toc122318424)

[3.6 Принцип обработки ошибок 22](#_Toc122318425)

[3.7 Структура и перечень сообщений лексического анализатора 22](#_Toc122318426)

[3.8 Параметры лексического анализатора и режимы его работы 23](#_Toc122318427)

[3.9 Алгоритм лексического анализа 23](#_Toc122318428)

[3.10 Контрольный пример 24](#_Toc122318429)

[Глава 4 Разработка синтаксического анализатора 25](#_Toc122318430)

[4.1 Структура синтаксического анализатора 25](#_Toc122318431)

[4.2 Контекстно свободная грамматика, описывающая синтаксис языка 25](#_Toc122318432)

[4.3 Построение конечного магазинного автомата 27](#_Toc122318433)

[4.4 Основные структуры данных 28](#_Toc122318434)

[4.5 Описание алгоритма синтаксического разбора 28](#_Toc122318435)

[4.6 Структура и перечень сообщений синтаксического анализатора 28](#_Toc122318436)

[4.7 Параметры синтаксического анализатора и режимы его работы 28](#_Toc122318437)

[4.8 Принцип обработки ошибок 29](#_Toc122318438)

[4.9 Контрольный пример 29](#_Toc122318439)

[Глава 5 Разработка семантического анализатора 30](#_Toc122318440)

[5.1 Структура семантического анализатора 30](#_Toc122318441)

[5.2 Функции семантического анализатора 30](#_Toc122318442)

[5.3 Структура и перечень сообщений семантического анализатора 30](#_Toc122318443)

[5.4 Принцип обработки ошибок 31](#_Toc122318444)

[5.5 Контрольный пример 31](#_Toc122318445)

[Глава 6 Преобразование выражений 32](#_Toc122318446)

[6.1 Выражения, допускаемые языком 32](#_Toc122318447)

[6.2 Польская запись и принцип ее построения 32](#_Toc122318448)

[6.3 Программная реализация обработки выражений 33](#_Toc122318449)

[6.4 Контрольный пример 33](#_Toc122318450)

[Глава 7 Генерация кода 34](#_Toc122318451)

[7.1 Структура генератора кода 34](#_Toc122318452)

[7.2 Представление типов данных в оперативной памяти 34](#_Toc122318453)

[7.3 Статическая библиотека 35](#_Toc122318454)

[7.4 Особенности алгоритма генерации кода 35](#_Toc122318455)

[7.5 Входные параметры, управляющие генерацией кода 35](#_Toc122318456)

[7.6 Контрольный пример 35](#_Toc122318457)

[Глава 8 Тестирование транслятора 36](#_Toc122318458)

[8.1 Общие положения 36](#_Toc122318459)

[8.2 Результаты тестирования 36](#_Toc122318460)

[Заключение 39](#_Toc122318461)

[Список использованных источников 40](#_Toc122318461)

[ПРИЛОЖЕНИЕ А 41](#_Toc122318462)

[ПРИЛОЖЕНИЕ Б 43](#_Toc122318463)

[ПРИЛОЖЕНИЕ В 45](#_Toc122318464)

[ПРИЛОЖЕНИЕ Г 47](#_Toc122318465)

[ПРИЛОЖЕНИЕ Д 49](#_Toc122318466)

[ПРИЛОЖЕНИЕ Е 52](#_Toc122318467)

# Введение

Задачей данного курсового проекта была поставлена разработка транслятора своего языка программирования KEM-2023. Он предназначен для работы с консолью, выполнения простейших действий, арифметических и побитовых операций над числами, а также для преобразования строк.

Главной задачей транслятора заключается в том, чтобы сделать исходный код на данном языке программирования понятной компьютеру. Для решения этой задачи был выбран способ трансляции исходного кода языка программирования в исходный код на языке ассемблера. Язык ассемблера — это машинно-ориентированный язык, представляющий формат записи машинных команд, которые понятны для восприятия человеком.

Компиляция состоит из двух частей: анализа и синтеза. Анализ — это разбиение исходной программы на составные части и создание ее промежуточного представления. Синтез — конструирование требуемой целевой программы из промежуточного представления. В данном курсовом проекте исходный код транслируется на язык ассемблера. Компилятор языка KEM-2023 состоит из следующих составных частей:

– лексический и семантический анализаторы;

– синтаксический анализатор;

– генератор исходного кода на языке ассемблера.

Исходя из цели курсового проекта, были определены следующие задачи:

– разработка спецификации языка программирования;

– разработка структуры транслятора;

– разработка лексического и семантического анализаторов;

– разработка синтаксического анализатора;

– преобразование выражений;

– генерация кода на язык ассемблера;

– тестирование транслятора.

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

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

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

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

В четвертой главе работы рассказывается о синтаксическом анализаторе, который выполняет синтаксический разбор текста с распечаткой протокола разбора и дерева разбора на основе таблицы лексем.

В пятой главе описан семантический анализатор, показана его работа (распечатка выданных сообщений в трёх примерах на разных этапах).

В шестой главе решены вопросы преобразования выражений, допускаемых языком и приведена часть протокола для контрольного примера, отображающая результаты преобразования выражений в польский формат.

В седьмой главе представлена генерация кода, где из промежуточного представления порождается код на целевом языке.

В восьмой главе описывается тестирование транслятора.

# 1 Спецификация языка программирования

**1.1 Характеристика языка программирования**

Язык программирования KEM-2023 является процедурным, строго типизированным, не объектно-ориентированным, компилируемым. Он транслируется в язык ассемблера в 2 этапа: сначала исходный код транслируется в байт-код (промежуточное представление), а затем происходит перевод байт-кода в язык ассемблера.

Процедурный язык программирования — язык высокого уровня, в котором используется метод разбиения программ на отдельные связанные между собой модули — подпрограммы (процедуры и функции). Программа на процедурном языке состоит из последовательности команд, определяющих процедуру решения задачи. Выполнение программы сводится к последовательному выполнению операторов.

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

Объектно-ориентированный язык программирования — язык, построенный на принципах объектно-ориентированного программирования. В основе концепции объектно-ориентированного программирования лежит понятие объекта — некой сущности, которая объединяет в себе поля (данные) и методы (выполняемые объектом действия).

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

KEM-2023 предназначен для работы с консолью, выполнения простейших действий, арифметических и побитовых операций над числами, а также для преобразования строк.

**1.2 Определение алфавита языка программирования**

Алфавит языка программирования — это фиксированный набор основных символов, из которых должен составляться любой текст на этом языке. С помощью этих символов могут быть записаны идентификаторы, выражения и операторы языка.

Язык программирования KEM-2023 включает в себя символы латинского алфавита верхнего и нижнего регистров, арабские цифры, знаки препинания, знаки арифметических и побитовых операций.

Алфавит языка KEM-2023 состоит из следующих множеств символов:

* латинские символы верхнего и нижнего регистра: {A,B,C,…,Z; a,b,c,…,z};
* цифры: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
* знаки арифметических и побитовых операций: {+, -, \*, /, %, &, |, ~};
* знаки-сепараторы и слова-сепараторы: {(), {}, [], ;, :, =, !, begin, end}.

**1.3 Применяемые сепараторы**

Сепараторы, используемые в языке программирования KEM-2023, приведены в таблице 1.1.

Таблица 1.1 — Применяемые сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение сепаратора |
| ; | Разделитель инструкций |
| { } | Программный блок |
| ( ) | Параметры, приоритетность операций |
| пробел | Служит для разделения программных конструкций. Допускается везде, кроме идентификаторов и ключевых слов |
| , | Разделитель параметров в функции |
| = | Оператор проверки на равенство. Используется в условных конструкциях |
| : | Оператор присваивания |
| +, -, \*, /, % | Арифметические операции |
| &, |, ~ | Побитовые операции |
| begin … end | Блок условной конструкции |

Символы-сепараторы — символы, используемые для разделения отдельных лексических единиц или функциональных элементов в исходном коде программы.

**1.4 Применяемые кодировки**

В основе алфавита KEM-2023 лежит таблица символов Windows-1251, которая представлена на рисунке 1.1.



Рисунок 1.1 — Таблица кодировки Windows-1251

Windows-1251 — набор символов и кодировка, являющаяся стандартной 8-битной кодировкой для русских версий Microsoft Windows до 10-й версии. Windows-1251 выгодно отличается от других 8‑битных кириллических кодировок (таких как CP866, KOI8-R и ISO 8859-5) наличием практически всех символов, использующихся в русской типографике для обычного текста (отсутствует только значок ударения); она также содержит все символы для других славянских языков: украинского, белорусского, сербского, македонского и болгарского.

**1.5 Типы данных**

В языке KEM-2023 допускается использование фундаментальных типов данных. Есть 3 типа данных: беззнаковый целый, логический и строковый. Пользовательские типы данных не поддерживаются.

Описание типов данных, предусмотренных в данным языке представлено в таблице 1.2.

Таблица 1.2 — Фундаментальные типы данных

|  |  |
| --- | --- |
| Тип данных | Описание |
| uninteger (беззнаковый целочисленный) | Фундаментальный тип данных. Используется для работы с числовыми значениями. В памяти занимает 2 байта.  Диапазон значений: 0 — 2 147 483 647.  Инициализация по умолчанию: 0.  Поддерживаемые операции:  + (бинарный) — оператор сложения;  - (бинарный) — оператор вычитания;  \* (бинарный) — оператор умножения;  / (бинарный) — оператор деления;  % (бинарный) — оператор остаток от деления;  & (бинарный) — оператор побитового И;  | (бинарный) — оператор побитового ИЛИ;  ~ (бинарный) — оператор инверсии;  : (бинарный) — оператор присваивания.  В составе условного оператора поддерживается следующая логическая операция:  = (бинарный) — оператор проверки на равенство. |
| boolean  (логический) | Фундаментальный тип данных. Используется для объявления логических переменных, которые принимают одно из двух значений — false или true (в таблице идентификаторов отображаются как 0 и 1 соответственно). В памяти занимает 1 байт. Инициализация по умолчанию: false. Операции над данными логического типа недоступны. |

Продолжение таблицы 1.2

|  |  |
| --- | --- |
| string (строковый) | Фундаментальный тип данных. Предусмотрен для объявления строк. Используется для работы с последовательностями символов (заключаемыми в двойные кавычки), каждый из которых занимает в памяти 1 байт.  Максимальное число символов в строке — 255.  Инициализация по умолчанию: строка нулевой длины.  Операции над данными строкового типа: присваивание строковому идентификатору строкового литерала или значения строковой функции, а также использование библиотечных функций. |

Фундаментальный тип данных — это тот, который представляет собой конкретную форму типа данных, и он вводится самим языком программирования. Следовательно, фундаментальный тип данных имеет свои собственные фундаментальные характеристики и свойства, определенные в языке.

**1.6 Преобразование типов данных**

В языке программирования KEM-2023 преобразование типов данных не поддерживается, так как данный язык является строго типизированным. Все типы данных определены однозначно и не могут быть преобразованы в другие.

**1.7 Идентификаторы**

Для именования функций, параметров и переменных используются идентификаторы. Не предусмотрены зарезервированные идентификаторы. Имя идентификаторов не должно совпадать с ключевыми словами языка и с именами функций стандартной библиотеки.

Общее количество идентификаторов ограничено максимальным размером таблицы идентификаторов. Идентификаторы должны начинаться только с символов латинского алфавита, могут содержать цифры. Идентификаторы, объявленные внутри функционального блока, получают префикс, идентичный имени функции, внутри которой они объявлены. Максимальная длина идентификатора (вместе с префиксом) равна 12 символам. В случае превышения заданной длины компилятор выдаёт ошибку о превышении длины идентификатора. Данные правила действуют для всех типов идентификаторов.

Имя идентификатора составляется по следующим образом:

* состоит из символов латинского алфавита [a…z] любого регистра и цифр;
* допускается использование цифр и нижнего подчёркивания (идентификатор не может начинаться с цифры).

**1.8 Литералы**

В языке KEM-2023 существует 3 типа литералов: целого, символьного и строкового типов. С помощью литералов осуществляется инициализация переменных. Краткое описание литералов рассматриваемого языка программирования представлено в таблице 1.3.

Таблица 1.3 — Описание литералов

|  |  |
| --- | --- |
| Тип литерала | Описание литерала |
| Строковые | Состоят из последовательности символов (букв, цифр, знаков препинания и т. д.), заключенной в двойные кавычки, инициализируются пустой строкой. |
| Целые | Последовательность цифр 0…9. Могут быть представлены как в десятичном, так и в восьмеричном (первые символы — 0o) представлении. |

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

**1.9 Объявления данных**

В языке программирования KEM-2023 переменная должны быть объявлена до ее использования. Областью видимости переменной является блок функции, в которой она определена. Вне блока функции определение функции запрещено. Не допустимо объявление глобальных переменных. Конструкция для объявления переменных:

make <тип данных> <идентификатор>;

Примеры объявления переменных на языке KEM-2023:

make boolean t;

make uninteger e;

make string str1;

Листинг 1.1

Для объявления функций используется ключевое слово process, перед которым указывается тип функции, после которого имя функции. Далее обязателен список параметров и тело функции.

Пример объявления функции на языке KEM-2023:

uninteger process func(uninteger a)

{

make uninteger sum;

sum : a + a;

out sum;

};

Листинг 1.2

Каждая переменная получает префикс — название функции, в которой она объявлена.

**1.10 Инициализация данных**

После объявлении переменной допускается инициализация данных. Объектами-инициализаторами могут быть идентификаторы, литералы, выражения и вызов функции. Переменной можно присвоить значение с помощью оператора присваивания (:). Он является бинарным, так что тут должно быть два операнда. Слева должна быть переменная, а справа выражение, дающее какое-либо значение. Конструкция для инициализации данных:

<идентификатор> : <литерал> | <идентификатор> | <выражение>;

Примеры инициализации переменных на языке KEM-2023:

t : 10;

str1 : "string1";

x : (func(t) + 4) - (15 & 16);

e : 15 | 5;

Листинг 1.3

Предусмотрены значения по умолчанию, если переменные не инициализированы: 0 – для целочисленных типов данных, пустая строка (строка размером 0) – для строкового типа данных, false – для логического типа данных.

**1.11 Инструкции языка**

В языке программирования KEM-2023 применяются инструкции, представленные в таблице 1.4.

Таблица 1.4 — Инструкции языка

|  |  |
| --- | --- |
| Инструкция языка | Синтаксис |
| Объявление переменной | make <тип данных> <идентификатор>; |
| Объявление функции | <тип\_данных> process <идентификатор> (<тип\_данных> <идентификатор>, …) {…}; |
| Вызов функции | <идентификатор\_функции>(<идентификатор>|<литерал>,…) |
| Присвоение значения | <идентификатор> : <литерал>|<идентификатор>|<выражение>; |
| Вывод данных | write(<идентификатор> | <литерал>); |
| Вывод данных с переходом на новую строку | writeline(<идентификатор>|<литерал>); |
| Возврат из функции | out <идентификатор>|<литерал>; |
| Условный оператор | if(<условие>) begin <тело\_оператора> end; |

Тело условного оператора не может быть пустым.

**1.12 Операции языка**

Язык программирования KEM-2023 может выполнять побитовые операции, представленные в таблице 1.5.

Таблица 1.5 — Приоритетности операций языка программирования KEM-2023

|  |  |
| --- | --- |
| Операция | Приоритетность |
| () | 1 |
| ~ (побитовая) | 5 |
| | (побитовая) | 2 |
| & (побитовая) | 3 |
| +; - (арифметические) | 4 |
| \*; /; % (арифметические) | 5 |

Операции языка применимы исключительно к целочисленному типу данных. Для строкового и логического типа операции языка не предусмотрены.

**1.13 Выражения и их вычисления**

Вычисление выражений — одна из важнейших задач любого языка программирования. Всякое выражение составляется согласно следующим правилам:

* Допускается использовать скобки для смены приоритета операций;
* Выражение записывается в строку без переносов;
* Использование двух подряд идущих операторов не допускается;
* Допускается использовать в выражении вызов функции.

В выражениях можно использовать все виды операций языка KEM-2023.

Булевские выражения могут содержать либо булевскую переменную, либо сравнение двух целочисленных значений. Вызов функции запрещён.

Круглые скобки в выражении используются для изменения приоритета операций. Также не допускается запись двух подряд идущих побитовых операций.

Перед генерацией кода каждое выражение приводится к записи в польской записи для удобства дальнейшего вычисления выражения на языке ассемблера.

Выражения можно использовать после операции присваивания и после оператора возвращения значения out.

**1.14 Конструкции языка**

Программа на языке KEM-2023 оформляется в виде функций пользователя и главной функции. При составлении функций рекомендуется выделять блоки и фрагменты и применять отступы для лучшей читаемости кода.

Ключевые программные конструкции языка программирования представлены в таблице 1.6.

Таблица 1.6 — Программные конструкции языка

|  |  |
| --- | --- |
| Конструкция | Представление в языке |
| Главная функция | Program  {  …  };  Является точкой входа в программу. |
| Функция | <тип\_данных> process <идентификатор> (<идентификатор>, …)  {  …  out <идентификатор> | <литерал> | <выражение>;  }; |
| Условный оператор | if(<условие>)  begin  <тело\_оператора>  end; |

**1.15 Область видимости идентификаторов**

Область видимости: сверху вниз. Переменные, объявленные в одной функции, недоступны в другой. Все объявления и операции с переменными происходят внутри какого-либо блока. Каждая переменная или параметр функции получают префикс – название функции, внутри которой они находятся.

Все идентификаторы являются локальными и обязаны быть объявленными внутри какой-либо функции. Глобальных переменных нет. Параметры видны только внутри функции, в которой объявлены. Объявление пользовательских областей видимости не предусмотрено.

**1.16 Семантические проверки**

Перечень семантических проверок, предусмотренных языком, приведен в таблице 1.7.

Таблица 1.7 — Перечень семантических проверок

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Идентификаторы функций не должны повторяться |
| 2 | Операнды в операторах ветвления и выхода из функции должны быть целочисленного типа |
| 3 | Тип данных передаваемых значений в функцию должен совпадать с типом параметров при её объявлении |
| 4 | Идентификатор должен быть объявлен до его использования. |
| 5 | Операнды в арифметическом выражении не могут быть различных типов |
| 6 | Тип возвращаемого функцией значения должен совпадать с типом функции |

Если семантическая проверка не проходит, то в файл журнала записывается соответствующая ошибка.

**1.17 Распределение оперативной памяти на этапе выполнения**

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

**1.18 Стандартная библиотека и её состав**

В языке KEM-2023 присутствует стандартная библиотека, которая реализована на языке C++. Содержимое библиотеки и описание функций представлено в таблице 1.8.

Таблица 1.8 — Стандартная библиотека языка KEM-2023

|  |  |
| --- | --- |
| Функция | Описание |
| string concat(string a, string b); | Строковая функция. Принимает в качестве параметров 2 строки. Копирует вторую строку к концу первой строки. |
| string copy(string a, string b); | Строковая функция. Принимает в качестве параметров 2 строки. Копирует содержимое второй строки в первую. |
| boolean comp(string a, string b); | Логическая функция. Принимает в качестве параметров 2 строки. Если строки равны, возвращает true, иначе false. |
| uninteger len(string a) | Функция принимает в качестве параметра строку и возвращает длину. |

Стандартная библиотека автоматически подключается при трансляции исходного кода.

**1.19 Ввод и вывод данных**

Вывод данных осуществляется с помощью ключевого слова write/writeline. На вывод можно передавать как непосредственно литералы, так и переменные, вместо которых будут выведены их значения.

Примеры вывода данных на языке KEM-2023:

writeline("Hello!");

write(x);

Листинг 1.4

Ввод данных не поддерживается языком программирования KEM-2023.

**1.20 Точка входа**

В языке KEM-2023 каждая программа должна содержать главную функцию (точку входа) Program, с первой инструкции которой начнётся последовательное выполнение команд программы. Наличие в исходном коде более одной точки входа недопустимо.

**1.21 Препроцессор**

Препроцессор, принимающий и выдающий некоторые данные на вход транслятору, в языке KEM-2023 отсутствует.

**1.22 Соглашения о вызовах**

В языке вызов функций происходит по соглашению о вызовах \_\_stdcall. Особенности \_\_stdcall:

– все параметры функции передаются через стек;

– память высвобождает вызываемый код;

– занесение в стек параметров идёт справа налево.

**1.23 Объектный код**

Исходный код языка транслируется сначала в язык ассемблера, а затем - в объектный код.

**1.24 Классификация сообщений транслятора**

Сообщения транслятора приведены в таблице 1.9.

Таблица 1.9 — Классификация сообщений транслятора

|  |  |
| --- | --- |
| Интервал | Описание ошибок |
| 0-1 | Системные ошибки |
| 100-112 | Ошибки параметров |
| 113-119 | Ошибки лексического анализа |
| 600-607 | Ошибки синтаксического анализа |
| 120-129 | Ошибки семантического анализа |

Генерируемые транслятором сообщения определяют степень его информативности, то есть сообщения транслятора должны давать максимально полную информацию о допущенной пользователем ошибке при написании программы.

**1.25 Контрольный пример**

Контрольный пример программы на языке KEM-2023 можно просмотреть в приложении А.

**Глава 2 Структура транслятора**

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

Транслятор преобразует программу, написанную на языке KEM-2023 в программу на языке ассемблера. Для указания выходных файлов используются входные параметры транслятора, которые описаны в пункте 2.2. Компонентами транслятора являются лексический, синтаксический и семантический анализаторы, а также генератор кода на язык ассемблера. Принцип работы представлен на рисунке 2.1.

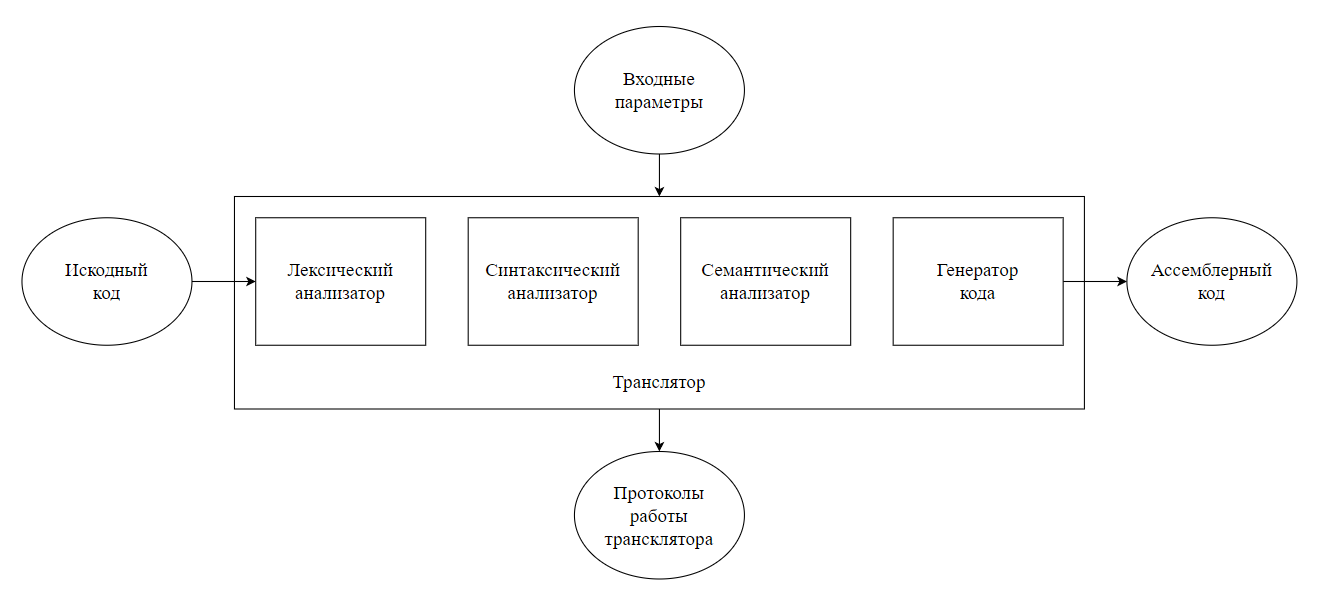


Рисунок 2.1 — Структура транслятора языка программирования KEM-2023

Первая стадия работы компилятора называется лексическим анализом, а программа, её реализующая, — лексическим анализатором (сканером). На вход лексического анализатора подаётся последовательность символов входного языка. Он производит предварительный разбор текста, преобразующий единый массив текстовых символов в массив отдельных слов (в теории компиляции вместо термина «слово» часто используют термин «токен»). Примеры лексических единиц: идентификаторы, числа, символы операций, служебные слова и т.д. Лексический анализатор преобразует исходный текст, заменяя лексические единицы их внутренним представлением — лексемами, для создания промежуточного представления исходной программы. Каждой лексеме сопоставляется ее тип и запись в таблице идентификаторов, в которой хранится дополнительная информация.

Таблица лексем (ТЛ) и таблица идентификаторов (ТИ) являются входом для следующей фазы компилятора — синтаксического анализа (разбора, или же парсера).

Цели лексического анализатора:

− убрать все лишние пробелы;

− выполнить распознавание лексем;

− построить таблицу лексем и таблицу идентификаторов;

− при неуспешном распознавании или обнаружении некоторых ошибок во входном тексте выдать сообщение об ошибке.

Синтаксический анализатор — часть компилятора, выполняющая синтаксический анализ, то есть проверку исходного кода на соответствие правилам грамматики. Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов. Выходной информацией является дерево разбора

Семантический анализ в свою очередь является проверкой исходной программы KEM-2023 на семантическую согласованность с определением языка, т.е. проверяет правильность текста исходной программы с точки зрения семантики.

Генератор кода — этап транслятора, выполняющий генерацию ассемблерного кода на основе полученных данных на предыдущих этапах трансляции. Генератор кода принимает на вход таблицы идентификаторов и лексем и транслирует код на языке KEM-2023, прошедший все предыдущие этапы, в код на языке Ассемблер.

**2.2 Перечень входных параметров транслятора**

Входные параметры для запуска консольного приложения представлены в таблице 2.1.

Таблица 2.1 — Входные параметры транслятора языка KEM-2023

|  |  |  |
| --- | --- | --- |
| Параметр | Назначение | Тип |
| -in: | Указывает на файл с исходным кодом программы на языке KEM-2023. Исходный код содержится в файле с расширением «.txt» | Обязательный |
| -out: | Указывает имена выходных файлов. Если не указан явно, то имена файлов со сгенерированным кодом формируются конкатенацией имени файла исходного кода и постфиксом «.asm». Также формируется вспомогательный файл с расширением «.html» | Необязательный |
| -log: | Указывает имя файла протокола. Если не указан явно, то имя протокола формируется конкатенацией имени файла исходного кода и постфикса «.log» | Необязательный |

В проекте KEM-2023 был использован только параметр -in.

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

Перечень протоколов, формируемых транслятором языка программирования KEM-2023, и их назначение представлены в таблице 2.2.

Таблица 2.2 — Протоколы, формируемые транслятором языка KEM-2023

|  |  |
| --- | --- |
| Формируемый протокол | Описание протокола |
| ".log" | В файле находится информация о входных параметрах, количестве символов исходного кода. При наличии ошибки выводится ее краткое описание и соответствующая ей позиция в исходном коде. |
| ".asm" | Результат работы программы — файл, содержащий исходный код на языке ассемблера. |
| ".idtable" | В файле находится таблица идентификаторов |
| ".lextable" | В файле находится таблица лексем |
| ".polish" | В файле находится таблица лексем в польской записи |

Протокол работы нужен для отображения хода выполнения трансляции языка KEM-2023. Благодаря им пользователь может обнаружить некорректно введенные данные или ошибки в исходном коде программы.

**Глава 3 Разработка лексического анализатора**

**3.1 Структура лексического анализатора**

Первая стадия работы компилятора называется лексическим анализом, а программа, её реализующая, — лексическим анализатором (сканером). На вход лексического анализатора подаётся последовательность символов входного языка. Лексический анализатор выделяет в этой последовательности простейшие конструкции языка, которые называют лексическими единицами. Примеры лексических единиц: идентификаторы, числа, символы операций, служебные слова и т. д. Лексический анализатор преобразует исходный текст, заменяя лексические единицы их внутренним представлением — лексемами, для создания промежуточного представления исходной программы. Каждому идентификатору и литералу в таблице лексем сопоставляется ее тип и запись в таблице идентификаторов, в которой хранится дополнительная информация. Структура лексического анализатора представлена на рисунке 3.1.

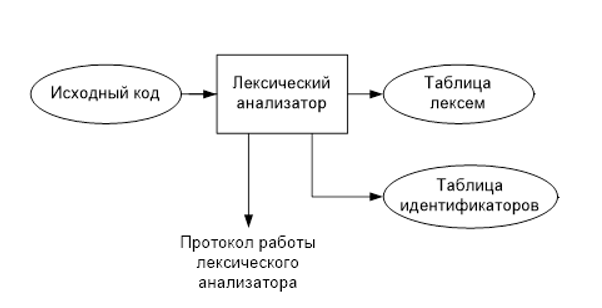


Рисунок 3.1 — Структура лексического анализатора KEM-2023

В результате формируется таблица лексем и таблица идентификаторов.

**3.2 Контроль входных символов**

Контроль входных символов в языках программирования представляет собой процесс проверки и валидации символов, вводимых пользователем или получаемых из внешних источников. Это важная задача, которая позволяет программе обрабатывать только допустимые символы и избегать ошибок или непредвиденного поведения.

Контроль входных символов может включать следующие аспекты: проверка типа символа, проверка наличия или отсутствия символов, проверка длины ввода, фильтрация символов, обработка ошибок.

Таблица для контроля входных символов представлена в листинге 3.1.

#define IN\_CODE\_TABLE {\

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

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

IN::P, IN::T, IN::T, IN::T, IN::T, IN::S, IN::S, IN::Q, IN::S, IN::S, IN::S, IN::S, IN::S, IN::S, IN::F, IN::S, \

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

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

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

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

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

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

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

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

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

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

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

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

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

Листинг 3.1

Принцип работы таблицы заключается в соответствии значения каждого элемента в шестнадцатеричной системе счисления значению в таблице кодировки Windows-1251.

Описание значения символов: T — разрешённый символ, F — запрещённый символ, S — сепаратор, P — пробелы, табуляция и переход на новую строку, Q — одинарная кавычка.

**3.3 Удаление избыточных символов**

Избыточными символами являются символы табуляции, символы конца строки, пробелы и символы, идущие после символа начала комментария (#), если он не включен в строковый литерал. Избыточные символы удаляются на этапе разбиения исходного кода на токены.

Описание алгоритма удаления избыточных символов:

1. Посимвольно считываем файл с исходным кодом программы.

2. Встреча пробела или знака табуляции является своего рода встречей символа-сепаратора.

3. В отличие от других символов-сепараторов не записываем в слово эти символы, только этот символ не является частью строкового литерала.

**3.4 Перечень ключевых слов**

Лексемы — это символы, соответствующие ключевым словам, символам операций и сепараторам, необходимые для упрощения дальнейшей обработки исходного кода программы. Данное соответствие описано в таблице 3.1.

Таблица 3.1 — Соответствие ключевых слов, символов операций и сепараторов с лексемами

|  |  |  |  |
| --- | --- | --- | --- |
| Тип цепочки | Цепочка | | Лексема |
| Ключевые слова | make | | m |
| uninteger | | t |
| string | | t |
| boolean | | t |
| process | | f |
| out | | o |
| write | | w |
| writeline | | w |
| Program | | p |
| if | | c |
| false | | t |
| true | | t |
| Иное | Идентификатор | | i |
| Литерал | | l |
| Сепараторы | ; | ; | |
| , | , | |
| Сепараторы | { | { | |
| } | } | |
| ( | ( | |
| ) | ) | |
| begin | [ | |
| end | ] | |
| Операторы | + | x | |
| - | x | |
| & | x | |
| | | x | |
| ~ | x | |
| = | x | |
| : | = | |

Пример реализации таблицы лексем представлен в приложении А. Также в приложении А находится таблица идентификаторов для контрольного примера.

**3.5 Основные структуры данных**

Основные структуры таблиц лексем и идентификаторов данных для языка KEM-2023, используемых для хранения, представлены в приложении А. В таблице лексем содержится лексема, её номер, полученный при разборе, номер строки в исходном коде, для арифметических операторов хранится их значение. В таблице идентификаторов содержится имя идентификатора, номер в таблице лексем, тип данных, смысловой тип идентификатора и его значение.

**3.6 Принцип обработки ошибок**

Если в результате работы лексического анализатора найдена ошибка, анализатор добавляет информацию в таблицу ошибок, после чего главная функция записывает ошибки в файл протокола. Они включают следующую информацию: код ошибки, номер строки в коде, номер позиции в строке или только код ошибки. Пользователь может ознакомиться со всеми ошибками, полученными в результате анализа, открыв протокол.

При возникновении ошибки работа транслятора прекращается.

**3.7 Структура и перечень сообщений лексического анализатора**

Структура сообщений содержит информацию о номере сообщения, номер строки, где было вызвано сообщение в исходном коде, а также информацию об ошибке.

Перечень сообщений об ошибках работы лексического анализатора представлен в таблице 3.2.

Таблица 3.2 — Перечень сообщений лексического анализатора

|  |  |
| --- | --- |
| Код | Сообщение |
| 113 | Элемент не распознан |
| 114 | Ошибка при создании файла IT |
| 115 | Ошибка при создании файла LT |
| 116 | Ошибка при создании лексической таблицы (превышен максимальный размер) |
| 117 | Ошибка при создании таблицы идентификаторов (превышен максимальный размер) |
| 118 | Ошибка при получении строки лексической таблицы (нет элемента) |
| 119 | Ошибка при получении строки таблицы идентификаторов (нет элемента) |

Все ошибки являются критическими и приводят к прекращению работы транслятора и выводу диагностического сообщения в log-файл.

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

Входным параметром лексического анализа являются редактированный код исходного файла, пустая таблица лексем, пустая таблица идентификаторов, таблица ошибок.

Лексический анализатор обрабатывает код исходного файла, проверяя символы и слова на соответствие возможным цепочкам, после чего заполняет таблицу лексем и таблицу идентификаторов.

**3.9 Алгоритм лексического анализа**

Алгоритм работы лексического анализа заключается в последовательном распознавании и разборе цепочек исходного кода и заполнение таблиц идентификаторов и лексем. Лексический анализатор производит распознаёт и разбирает цепочки исходного текста программы. Это основывается на работе конечных автоматов, которую можно представить в виде графов. В случае, если подходящий автомат не был обнаружен, запоминается номер строки, в которой находился этот токен и выводится сообщение об ошибке. Если токен разобран, то дальнейшие действия, которые будут с ним производиться, будут зависеть от того, чем он является. Регулярные выражения — аналитический или формульный способ задания регулярных языков. Они состоят из констант и операторов, которые определяют множества строк и множество операций над ними. Любое регулярное выражение можно представить в виде графа.

В случае, если токен является идентификатором, перед его именем записывается название функции, в которой он объявлен и после этого он заносится в таблицу идентификаторов.

В случае, если токен является идентификатором функции, название функции в которой он объявлен не записывается.

В случае, если токен является литералом, то он заносится в таблицу идентификаторов в виде ln, где b — “L”, n — текущий индекс литерала.

Когда встречаем токен, являющийся ключевым словом, которое отвечает за тип данных или вид идентификатора, заносим лексему, соответствующую ему, в таблицу лексем и запоминаем тип данных или вид идентификатора, которому он соответствует.

В последствии, когда встречаем идентификатор, заносим его в таблицу идентификаторов с соответствующим ему типом данных и видом идентификатора, и именем вида “i\_f”, где i — имя идентификатора, f — имя функции, где объявлен идентификатор.

Граф КА для ключевого слова make представлен на рисунке 3.4.

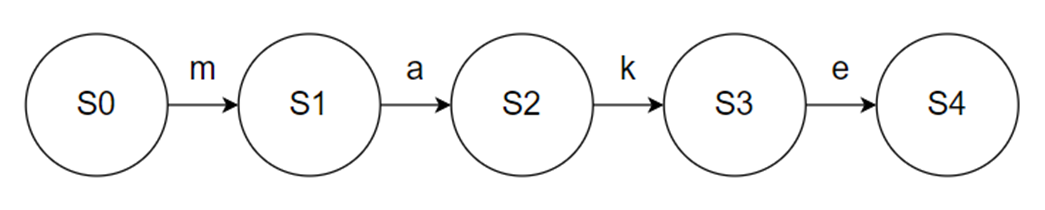


Рисунок 3.2 — Граф переходов для цепочки «make»

Здесь S0 — начальное состояние, S4 — конечное состояние автомата.

**3.10 Контрольный пример**

Результат работы лексического анализатора представлен в виде таблицы лексем и таблицы идентификаторов в приложении А.

**Глава 4 Разработка синтаксического анализатора**

**4.1 Структура синтаксического анализатора**

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



Рисунок 4.1 — Структура синтаксического анализатора

Входом для синтаксического анализа является таблица лексем и таблица идентификаторов, полученные после фазы лексического анализа, выходом — дерево разбора.

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

В синтаксическом анализаторе транслятора языка KEM-2023 используется контекстно-свободная грамматика *G = <T, N, P, S>*, 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 нетерминалом.

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

1. *A → aα*, где *a ∈ T*, *α* ∈ *(T* ∪*N)* ∪ *λ*; (или *α* ∈ *(T* ∪*N)\**, или *α* ∈ *V\**)
2. *S→ λ*, где *S* ∈ *N* — начальный символ, при этом если такое правило существует, то нетерминал S не встречается в правой части правил.

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

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

Таблица 4.1 — Перечень правил, составляющих грамматику языка и описание нетерминальных символов KEM-2023

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| S | p{N};  tfi(F){NoR;};S  tfi(F){oR;};S | Порождает правила, описывающие общую структуру программы |
| N | mti;  i=E;  mti;N  i=E;N  mtfi(F);N  w(R);  w(R);N  c(I)[N];N  c(I)[N];  fi(F);N};  i(W);N  oR; | Порождает правила, описывающие инструкции языка |
| E | i  l  (E)  i()  i(W)  iM  lM  (E)M  i()M  i(W)M | Порождает правила, описывающие выражения |
| M | xE  xEM | Порождает правила, описывающие побитовые и арифметические операции |
| F | ti  ti,F | Порождает правила, описывающие параметры шаблона функции |
| W | i  l  i,W  l,W | Порождает правила, описывающие параметры функции |
| R | l  i | Порождает правила, описывающие возвращаемый тип и операнды вывода в консоль |
| I | ixi  lxl  lxi  ixl  i  l | Порождает правила, описывающие условную конструкцию |

Правила языка KEM-2023 представлена в приложении Б.

**4.3 Построение конечного магазинного автомата**

Конечный автомат с магазинной памятью представляет собой семерку*M=<Q,V,Z,δ,q0,z0,F>*, описание которой представлено в таблице 4.2.

Таблица 4.2 – Описание компонентов магазинного автомата

|  |  |  |
| --- | --- | --- |
| Компонента | Определение | Описание |
| *Q* | Множество состояний автомата | Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата |
| *V* | Алфавит входных символов | Алфавит является множеством терминальных и нетерминальных символов, описание которых содержится в разделе 1.2 и в таблице 4.1. |
| *Z* | Алфавит специальных символов | Алфавит магазинных символов содержит стартовый символ и маркер дна стека |
| *δ* | Функция переходов автомата | Функция представляет из себя множество правил грамматики, описанных в таблице 4.1. |
| *q0* | Начальное состояние автомата | Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики (нетерминальный символ S) |
| *z0* | Начальное состояние автомата | Символ маркера дна стека ($) |
| *F* | 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 |

Пример работы магазинного автомата для цепочки представлен на риc. 4.2.

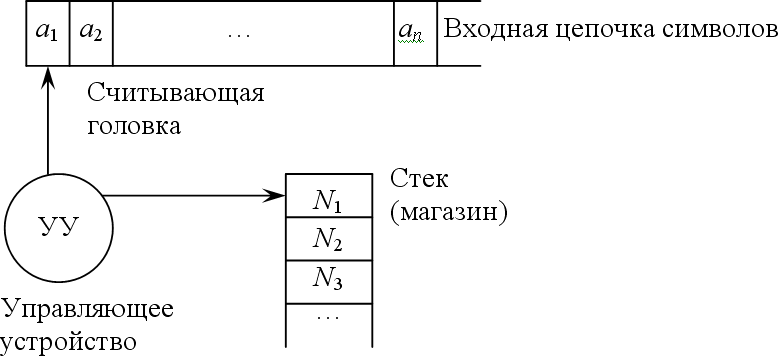


Рисунок 4.2 — Автомат с магазинной памятью

Структура магазинного автомата и функции, реализованные для его работы, прокомментированы показаны в приложении В.

**4.4 Основные структуры данных**

Основные структуры данных синтаксического анализатора представляются в виде структуры магазинного конечного автомата, выполняющего разбор исходной ленты, и структуры грамматики Грейбах, описывающей синтаксические правила языка KEM-2023. Данные структуры представлены в приложении В.

**4.5 Описание алгоритма синтаксического разбора**

Принцип работы автомата следующий:

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

**4.6 Структура и перечень сообщений синтаксического анализатора**

Перечень сообщений синтаксического анализатора представлен в таблице 4.3.

Таблица 4.3 — Перечень сообщений синтаксического анализатора

|  |  |
| --- | --- |
| Код | Сообщение |
| 600 | Неверная структура программы |
| 601 | Ошибочный оператор или функция не возвращает значение |
| 602 | Ошибка в выражениях |
| 603 | Ошибка в операторах выражения |
| 604 | Ошибка в параметрах функции |
| 605 | Ошибка в параметрах функции |
| 606 | Значением данного оператора может быть только переменная или литерал |
| 607 | Ошибка в условии условного оператора |

Количество правил соответствует количеству ошибок.

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

Входным параметром синтаксического анализатора является таблица лексем, полученная на этапе лексического анализа, а также правила контекстно-свободной грамматики в форме Грейбах.

Выходными параметрами являются трассировка прохода таблицы лексем (при наличии разрешающего ключа) и правила разбора, которые выводятся в консоль.

**4.8 Принцип обработки ошибок**

Обработка ошибок происходит следующим образом:

1. Синтаксический анализатор перебирает все правила и цепочки правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем.
2. Если невозможно подобрать подходящую цепочку, то генерируется соответствующая ошибка.
3. Все ошибки записываются в общую структуру ошибок.
4. В случае нахождения ошибки, после всей процедуры трассировки в протокол будет выведено диагностическое сообщение.

**4.9 Контрольный пример**

Пример разбора исходного кода на языке KEM-2023 синтаксическим анализатором представлен в приложении Г.

**Глава 5 Разработка семантического анализатора**

**5.1 Структура семантического анализатора**

Семантический анализатор принимает на свой вход результаты работ лексического и синтаксического анализаторов, то есть таблицы лексем, идентификаторов и результат работы синтаксического анализатора, то есть дерево разбора, и последовательно ищет необходимые ошибки. Некоторые проверки (такие как проверка на единственность точки входа, проверка на предварительное объявление переменной) осуществляются в процессе лексического анализа. Общая структура обособленно работающего (не параллельно с лексическим анализом) семантического анализатора представлена на рисунке 5.1.



Рисунок 5.1 — Структура семантического анализатора

**5.2 Функции семантического анализатора**

Семантический анализатор выполняет проверку на основе правил языка, описанных в пункте 1.16. Он и есть та самая подпрограмма, которая занимается автоматическим приведением типов.

**5.3 Структура и перечень сообщений семантического анализатора**

Сообщения, формируемые семантическим анализатором, представлены в таблице 5.1.

Таблица 5.1 — Перечень сообщений семантического анализатора

|  |  |
| --- | --- |
| Код | Сообщение |
| 120 | Невозможно применить оператор к данному типу операнда |
| 121 | Тип возвращаемого значения не соответствует типу функции |
| 122 | Использована необъявленная переменная |
| 123 | Повторное объявление имени |
| 124 | Функция должна возвращать значение |

Продолжение таблицы 5.1

|  |  |
| --- | --- |
| Код | Сообщение |
| 125 | Ошибка в условии условного оператора |
| 126 | Длина строкового литерала превышает допустимое значение |
| 127 | Ошибка в условии условного оператора |
| 128 | Несовпадение фактических и формальных параметров функции |
| 129 | Обнаружено больше 1 точки входа в программу |

**5.4 Принцип обработки ошибок**

Семантический анализатор в случае возникновения ошибки заносит её в протокол. Следующий этап трансляции не будет запущен при возникновении ошибки. Семантический анализ начинает проверки уже на стадии лексического анализа, если на этой стадии обнаружены семантические ошибки — программа завершит свою работу.

**5.5 Контрольный пример**

Демонстрации ошибок, диагностируемых семантическим анализатором на разных этапах трансляции приведена в разделе 8.2.

# Глава 6 Преобразование выражений

**6.1 Выражения, допускаемые языком**

В языке KEM-2023 допускаются выражения, применимые к целочисленным типам данных. В выражениях поддерживаются арифметические операции (сложение, вычитание, умножение, деление и нахождение остатка от деления), побитовые операции (и, или и инверсия) и вызовы функций как операнды арифметических выражений.

Приоритет операций представлен в таблице 6.1.

Таблица 6.1 — Приоритет операций в языке KEM-2023

|  |  |
| --- | --- |
| Операция | Приоритетность |
| () | 1 |
| ~ (побитовая) | 5 |
| | (побитовая) | 2 |
| & (побитовая) | 3 |
| +; - (арифметические) | 4 |
| \*; /; % (арифметические) | 3 |

Выражения и операции, допускаемые языком KEM-2023, подробно описаны в разделах 1.12 и 1.13.

**6.2 Польская запись и принцип ее построения**

Выражения в языке KEM-2023 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:

1. Исходная строка: выражение.
2. Результирующая строка: польская запись.
3. Стек: пустой.
4. Исходная строка просматривается слева направо.
5. Операнды переносятся в результирующую строку в порядке их следования.
6. Операция записывается в стек, если стек пуст или в вершине стека лежит отрывающая скобка.
7. Операция выталкивает все операции с большим или равным приоритетом в результирующую строку.
8. Запятая не помещается в стек, если в стеке операции, то все выбираются в строку.
9. Отрывающая скобка помещается в стек.
10. Закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются.
11. Закрывающая скобка, если поднят флаг функции, выталкивает все до открывающей скобки в обратном порядке и добавляет идентификатор функции в конце.
12. По концу разбора исходной строки все операции, оставшиеся в стеке, выталкиваются в результирующую строку.

Пример преобразования выражения в обратную польскую запись представлен в таблице 6.2.

Таблица 6.2 — Пример преобразования выражения в обратную польскую запись

|  |  |  |
| --- | --- | --- |
| Исходная строка | Результирующая строка | Стек |
| (a+b) - f(i) |  |  |
| +b) - f(i) | a | ( |
| b) - f(i) | a | (+ |
| ) - f(i) | ab | (+ |
| - f(i) | ab+ |  |
| f(i) | ab+c\* | - |
|  | ab+i@1- |  |

**6.3 Программная реализация обработки выражений**

Программная реализация алгоритма преобразования выражений к польской записи с комментариями представлена в приложении Д.

**6.4 Контрольный пример**

Пример преобразования выражения к обратной польской записи представлен в таблице 6.2. Преобразование выражений в формат польской записи необходимо для построения более простых алгоритмов их вычисления.

**Глава 7 Генерация кода**

**7.1 Структура генератора кода**

Заключительным этапом трансляции языка KEM-2023 является генерация кода. Генератор принимает на вход таблицу лексем, таблицу идентификаторов и дерево разбора. На выходе получается файл с исходным кодом на ассемблере, который будет являться результатом работы транслятора. В случае возникновения ошибок генерация кода не будет осуществляться. Структура генератора кода представлена на рисунке 7.1.



Рисунок 7.1 – Структура генератора кода

Генератор кода начинает свою работу только в том случае, если код прошёл предыдущие этапы без ошибок.

**7.2 Представление типов данных в оперативной памяти**

Элементы таблицы идентификаторов расположены в разных сегментах языка ассемблера – .data и .const. Идентификаторы языка KEM-2023 размещены в сегменте данных(.data). Литералы – в сегменте констант (.const). Соответствия между типами данных идентификаторов на языке KEM-2023 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствие типов идентификаторов языка KEM-2023 и языка Ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке KEM-2023 | Тип идентификатора на языке ассемблера | Пояснение |
| uninteger  boolean | DWORD | Хранит целочисленный тип данных (для boolean: 0 или 1). |
| string | DWORD | Хранит указатель на начало строки. |
| Литерал | BYTE  DWORD | Литералы: строковые,  целочисленные |

Сгенерированный код на языке Assembler приведен в приложении Е.

**7.3 Статическая библиотека**

В языке KEM-2023 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++ и автоматически подключается на начальном этапе генерации кода. Все функции статической библиотеки описаны в разделе 1.18.

**7.4 Особенности алгоритма генерации кода**

В языке KEM-2023 генерация кода строится на основе таблиц лексем и идентификаторов. Общая схема работы генератора кода представлена на рисунке 7.2.

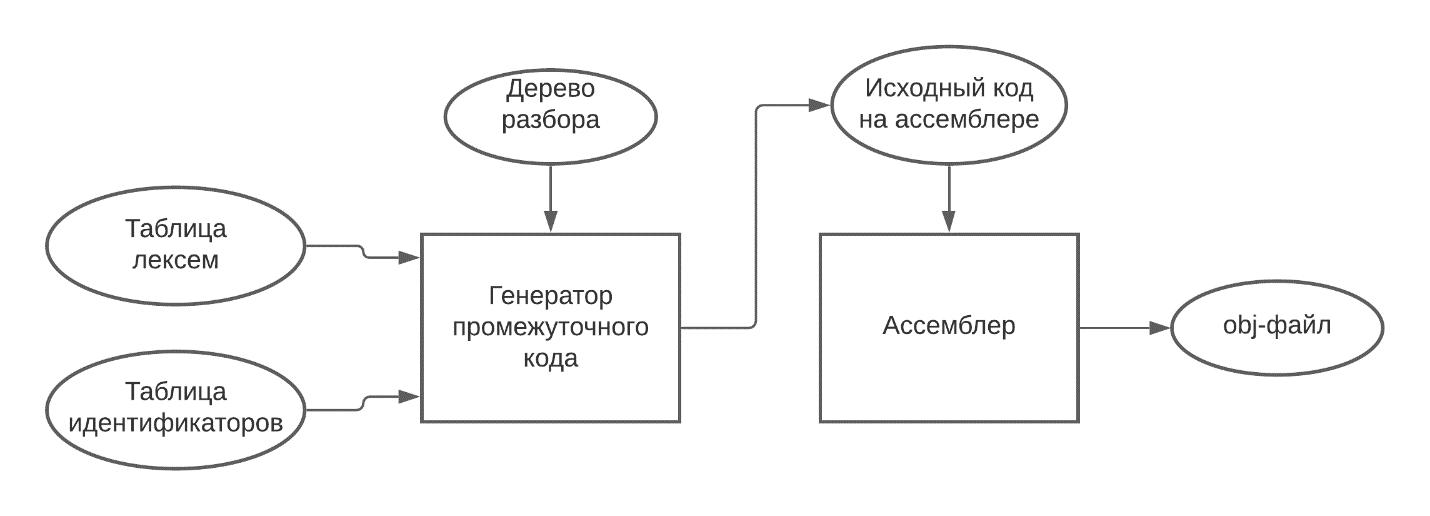


Рисунок 7.2 — Структура генератора кода

**7.5 Входные параметры, управляющие генерацией кода**

На вход генератору кода поступают таблицы лексем и идентификаторов исходного кода программы на языке KEM-2023. Результат работы генератора кода выводятся в файл с расширением .asm.

**7.6 Контрольный пример**

Результат генерации ассемблерного кода на основе контрольного примера из приложения А приведен в приложении Е.

**Глава 8 Тестирование транслятора**

**8.1 Общие положения**

В результате обработки исходного кода программы, представленного в приложении А, транслятор языка KEM-2023 генерирует общий протокол работы, куда записываются все возникшие ошибки и предупреждения. Кроме того, все ошибки, возникшие на этапах лексического и семантического анализов, выводятся на консоль. Из ошибок, возникших на этапе синтаксического анализа, на консоль выводится только первая.

**8.2 Результаты тестирования**

В таблице 8.1 приведены ошибки, генерируемые в процессе считывания входного файла, а также в процессе лексического, синтаксического и семантического анализов.

Таблица 8.1 — Результаты тестирования транслятора

|  |  |
| --- | --- |
| Исходный код | Генерируемая ошибка |
| Program {  out 0;  } | Ошибка 600: Синтаксическая ошибка. Неверная структура программы |
| Program {  make uninteger new;  new = восемь;  out new;  }; | Ошибка 111: Недопустимый символ в исходном файле (-in) |
| uninteger process func(uninteger abc) {  make uninteger sum;  sum : abc + abc;  out mul;  }; | Ошибка 122: Семантическая ошибка. Использована необъявленная переменная |
| Program {  make boolean t;  t : true;  make boolean t;  out t;  }; | Ошибка 123: Семантическая ошибка. Повторное объявление имени |
| uninteger process func(uninteger abc) {  make uninteger sum;  sum : abc + abc;  out "str";  }; | Ошибка 121: Семантическая ошибка. Тип возвращаемого значения не соответствует типу функции |

Продолжение таблицы 8.1

|  |  |
| --- | --- |
| uninteger process func(uninteger abc) {  make string sum;  sum : abc + abc;  out sum;  }; | Ошибка 127: Семантическая ошибка. Несоответствие типов в выражении |
| uninteger process func(uninteger abc) {  out abc;  };  Program {  make uninteger x;  x : (func("k") + 4) - (15 & 16);  writeline(x);  out 0;  }; | Ошибка 128: Семантическая ошибка. Несовпадение фактических и формальных параметров функции |
| if(y)  begin  make uninteger s;  s : len(str1);  if(s = "hi")  begin  writeline(s);  end;  writeline(str2);  end; | Ошибка 125: Семантическая ошибка. Ошибка в условии условного оператора |
| uninteger process func(string boolean abc)  {  make uninteger sum;  sum : abc + abc;  out sum;  }; | Ошибка 604: Синтаксическая ошибка. Ошибка в параметрах функции |
| uninteger process func(uninteger abc) {  out abc;  };  Program {  make uninteger x;  x : 5;  writeline(x);  out func(x);  }; | Ошибка 606: Синтаксическая ошибка. Значением данного оператора может быть только переменная или литерал |

Окончание таблицы 8.1

|  |  |
| --- | --- |
| Program {  make string sum;  sum : abc + abc;  }; | Ошибка 124: Cемантическая ошибка. Функция должна возвращать значение |
| if(y, t)  begin  make uninteger s;  s : len(“1234567”);  if(s = 7)  begin  writeline(s);  end;  writeline(“stop”);  end; | Ошибка 607: Синтаксическая ошибка. Ошибка в условии условного оператора |
| uninteger process func(uninteger abc) {  out abc;  };  Program {  make uninteger x;  x : (func(5) + 4), - (15 & 16);  writeline(x);  out 0;  }; | Ошибка 603: Синтаксическая ошибка. Ошибка в операторах выражений |
| uninteger process func(uninteger abc) {  out abc;  };  Program {  make uninteger x;  x : (func(5) + 4) - (make);  writeline(x);  out 0;  }; | Ошибка 602: Синтаксическая ошибка. Ошибка в выражении |

**Заключение**

В ходе проделанной работы был разработан транслятор для языка программирования KEM-2023.

Таким образом, были выполнены основные задачи данной курсовой работы:

1. сформулирована спецификация языка KEM-2023;
2. разработаны конечные автоматы и алгоритмы для реализация лексического анализатора;
3. разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
4. разработан семантический анализатор, осуществляющий проверку смысла используемых инструкций;
5. разработан транслятор с языка программирования KEM-2023 на язык Ассемблер;
6. проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка KEM-2023 включает:

* 5 арифметических операторов и 3 побитовых оператора для вычисления выражений;
* 3 типа данных;
* оператор вывода;
* подключаемую стандартную библиотеку;
* более 2000 строк кода;
* функции лексического сравнения, конкатенации, копирования и вычисления длины строк;
* условную конструкцию;
* арифметику целочисленных литералов;
* структурированную систему для обработки ошибок.

Таким образом была достигнута поставленная цель по разработке компилятора KEM-2023, были учтены все требования, все задачи курсового проекта выполнены.

# Список использованных источников

1. Карпов Ю. Теория и технология программирования. Основы построения трансляторов, 2005. – 272с.
2. Введение в теорию трансляторов [Электронный ресурс]. – Режим доступа: http://bourabai.ru/tpoi/compilers.htm. – Дата доступа: 30.10.2023.
3. Википедия: Обратная польская запись [Электронный ресурс]. – Режим доступа: https://en.wikipedia.org/wiki/Reverse\_Polish\_notation. – Дата доступа: 12.11.2023.
4. MASM для x86 [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/en-us/cpp/assembler/masm/masm-for-x64-ml64-exe?view=msvc-160>. – Дата доступа: 27.11.2023.
5. Ирвин К. Р. Язык ассемблера для процессоров Intel / К. Р. Ирвин. – M.: Вильямс, 2005. – 912с.

# ПРИЛОЖЕНИЕ А

|  |
| --- |
| uninteger process func(uninteger abc)  {  make uninteger sum;  sum : abc + abc;  out sum;  };  Program  {  make boolean t;  t : true;  if (t)  begin  writeline("Hello");  end;  make uninteger e;  e : 15 | 5;  make uninteger x;  x : (func(g) + 4) - (15 & 16);  writeline(x);  make string str1;  make string str2;  str1 : "string1";  str2 : "string2";  make string str3;  str3 : concat(str1, str2);  writeline(str3);  make boolean y;  y : true;  if(y)  begin  make uninteger s;  s : len(str1);  if(s = 7)  begin  writeline(s);  end;  writeline(str2);  end;  make uninteger f;  f : 0o123;  out 0;  }; |

Листинг A.1 — Контрольный пример

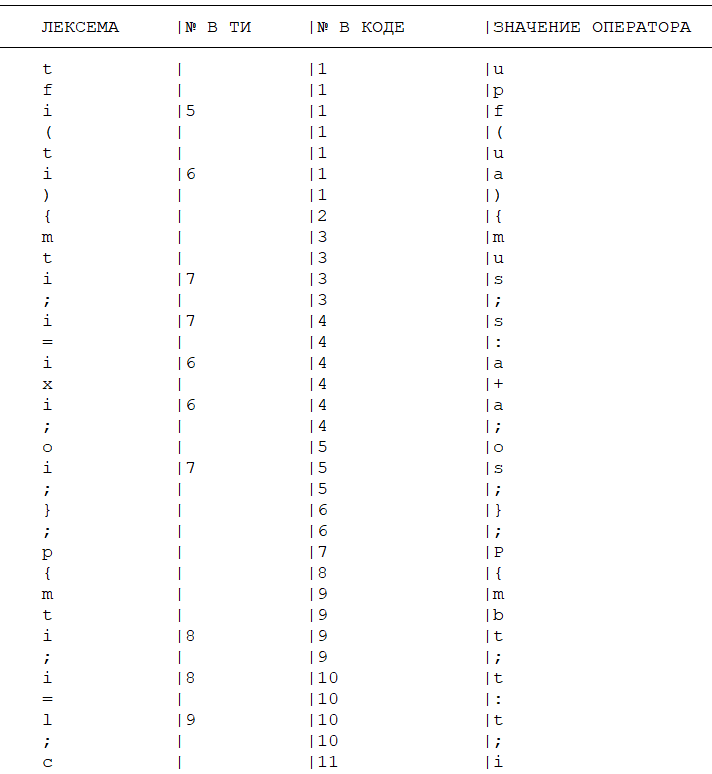
****

Рисунок A.1 — Часть таблицы лексем

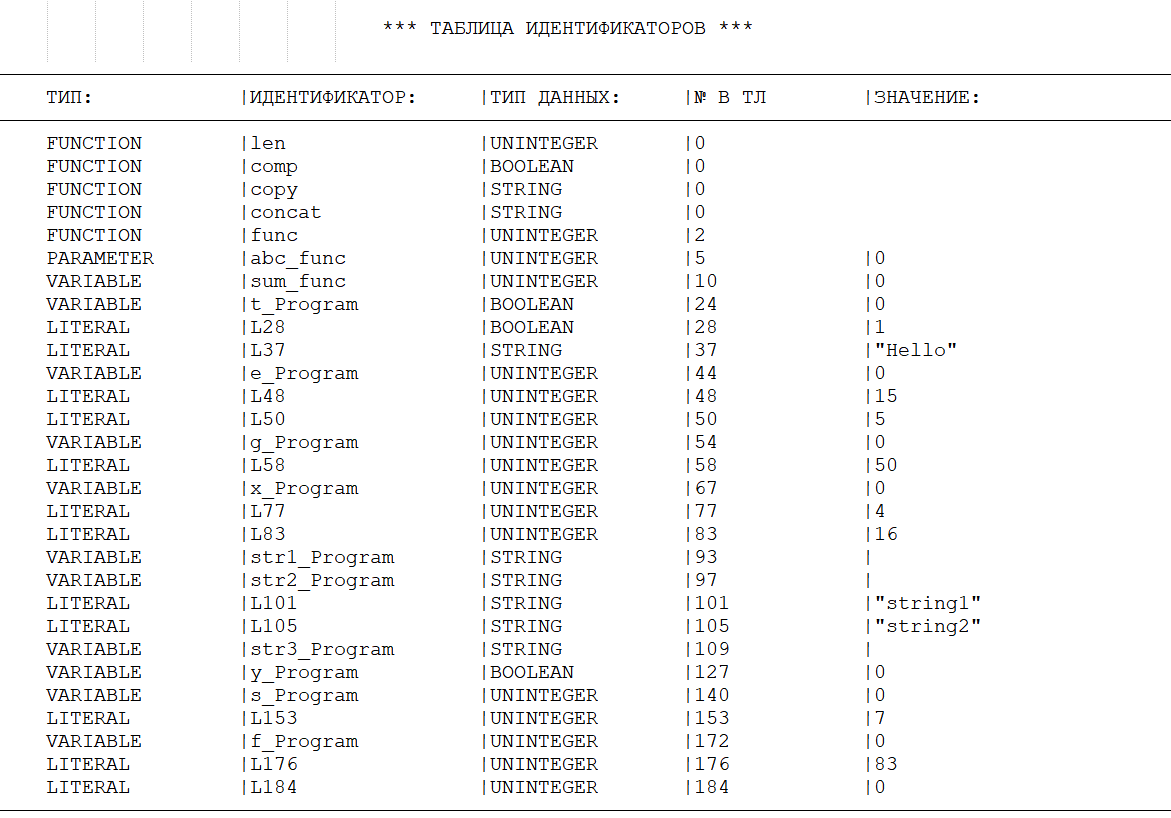
****

Рисунок A.2 — Таблица идентификаторов

# ПРИЛОЖЕНИЕ Б

|  |
| --- |
| Greibach greibach(NS('S'), TS('$'), 8,  Rule(NS('S'), GRB\_ERROR\_SERIES + 0, 3, // Неверная структура программы  Rule::Chain(5, TS('p'), TS('{'), NS('N'), TS('}'), TS(';')),  Rule::Chain(14, TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), NS('N'), TS('o'), NS('R'), TS(';'), TS('}'), TS(';'), NS('S')),  Rule::Chain(13, TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS('{'), TS('o'), NS('R'), TS(';'), TS('}'), TS(';'), NS('S'))  ),  Rule(NS('N'), GRB\_ERROR\_SERIES + 1, 12, // Ошибочный оператор  Rule::Chain(4, TS('m'), TS('t'), TS('i'), TS(';')),  Rule::Chain(4, TS('i'), TS('='), NS('E'), TS(';')),  Rule::Chain(5, TS('m'), TS('t'), TS('i'), TS(';'), NS('N')),  Rule::Chain(5, TS('i'), TS('='), NS('E'), TS(';'), NS('N')),  Rule::Chain(9, TS('m'), TS('t'), TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS(';'), NS('N')),  Rule::Chain(5, TS('w'), TS('('), NS('R'), TS(')'), TS(';')),  Rule::Chain(6, TS('w'), TS('('), NS('R'), TS(')'), TS(';'), NS('N')),  Rule::Chain(9, TS('c'), TS('('), NS('I'), TS(')'), TS('['), NS('N'), TS(']'), TS(';'), NS('N')),  Rule::Chain(8, TS('c'), TS('('), NS('I'), TS(')'), TS('['), NS('N'), TS(']'), TS(';')),  Rule::Chain(9, TS('f'), TS('i'), TS('('), NS('F'), TS(')'), TS(';'), NS('N'), TS('}'), TS(';')),  Rule::Chain(5, TS('i'), TS('('), NS('W'), TS(')'), TS(';'), NS('N')),  Rule::Chain(3, TS('o'), NS('R'), TS(';'))  ),  Rule(NS('E'), GRB\_ERROR\_SERIES + 2, 10, // Ошибка в выражении  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l')),  Rule::Chain(3, TS('('), NS('E'), TS(')')),  Rule::Chain(3, TS('i'), TS('('), TS(')')),  Rule::Chain(4, TS('i'), TS('('), NS('W'), 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(4, TS('i'), TS('('), TS(')'), NS('M')),  Rule::Chain(5, TS('i'), TS('('), NS('W'), TS(')'), NS('M'))  ),  Rule(NS('M'), GRB\_ERROR\_SERIES + 3, 2,  Rule::Chain(2, TS('x'), NS('E')),  Rule::Chain(3, TS('x'), NS('E'), NS('M'))  ),  Rule(NS('F'), GRB\_ERROR\_SERIES + 4, 2, // Ошибка в параметрах  Rule::Chain(2, TS('t'), TS('i')),  Rule::Chain(4, TS('t'), TS('i'), TS(','), NS('F'))  ),  Rule(NS('W'), GRB\_ERROR\_SERIES + 5, 4, // Ошибка в пар-ах вызыв. ф-ции  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('R'), GRB\_ERROR\_SERIES + 6, 2, // Только переменная или литерал  Rule::Chain(1, TS('i')),  Rule::Chain(1, TS('l'))  ),  Rule(NS('I'), GRB\_ERROR\_SERIES + 7, 6, // Ошибка в условном операторе  Rule::Chain(1, TS('l')),  Rule::Chain(1, TS('i')),  Rule::Chain(3, TS('i'), TS('x'), TS('i')),  Rule::Chain(3, TS('i'), TS('x'), TS('l')),  Rule::Chain(3, TS('l'), TS('x'), TS('i')),  Rule::Chain(3, TS('l'), TS('x'), TS('l'))  )  ); |

Листинг Б.1 — Правила, описывающие грамматику языка

# ПРИЛОЖЕНИЕ В

|  |
| --- |
| struct MfstState { // состояние автомата (для сохранения)  short lenta\_position; // позиция на ленте  short nrule; // номер текущего правила  short nrulechain; // номер текущей цепочки, текущего правила  MFSTSTACK st; // стек автомата  MfstState();  MfstState(  short pposition, // позиция на ленте  MFSTSTACK pst, // стек автомата  short pnrulechain // номер текущей цепочки, текущего правила  );  MfstState(  short pposition, // позиция на ленте  MFSTSTACK pst, // стек автомата  short pnrule, // номер текущего правила  short pnrulechain // номер текущей цепочки, текущего правила  );  };  struct Mfst { // магазинный автомат  enum RC\_STEP { // код возврата функции step  NS\_OK, // нашли правило и цепочку, цепочка -> в стек  NS\_NORULE, // не найдено правило (ошибка в грамматике)  NS\_NORULECHAIN, // не найдена цепочка (ошибка в исходном коде)  NS\_ERROR, // неизвестный нетерм. символ грамматики  TS\_OK, // тек. символ = вершине стека, продв. лента  TS\_NOK, // тек. символ != вершине стека, восст. состояние  LENTA\_END, // текущая позиция ленты >= lenta\_size  SURPRISE // неожиданный код возврата (ошибка в step)  };  struct MfstDiagnosis { // диагностика  short lenta\_position; // позиция на ленте  RC\_STEP rc\_step; // код завершения шага  short nrule; // номер правила  short nrule\_chain; // номер цепочки правила  MfstDiagnosis();  MfstDiagnosis(  short plenta\_position, // позиция на ленте  RC\_STEP prt\_step, // код завершения шага  short pnrule, // номер правила  short pnrule\_chain // номер цепочки правила  );  } diagnosis[MFST\_DIAGN\_NUMBER]; // последние самые глубокие сообщения  GRBALPHABET\* lenta; // перекодированная (TS/NS) лента  short lenta\_position; // текущая позиция на ленте  short nrule; // номер текущего правила  short nrulechain; // номер тек. цепочки, тек. правила  short lenta\_size; // размер ленты  GRB::Greibach grebach; // грамматика Грейбах  LA::Tables lex; // результат работы лекс. анализатора  MFSTSTACK st; // стек автомата  std::stack<MfstState> storestate; // стек для сохранения состояний  Mfst();  Mfst(  LA::Tables pt, // результат работы лекс. анализатора  GRB::Greibach pgrebach // грамматика Грейбах  );  char\* getCSt(char\* buf); // получить содержимое стека  char\* getCLenta(char\* buf, short pos, short n = 25);  // лента: n символов с pos  char\* getDiagnosis(short n, char\* buf);  // получить n-ю строку диагностики или 0х00  bool savestate(); // сохранить состояние автомата  bool restate(); // восстановить состояние автомата  bool push\_chain( // поместить цепочку правила в стек  GRB::Rule::Chain chain // цепочка правила  );  RC\_STEP step(); // выполнить шаг автомата  bool start(); // запустить автомат  bool savediagnosis(  RC\_STEP pprc\_step // код завершения шага  );  void printrules(); // вывести последовательность правил  struct Deducation { // вывод  short size; // количество шагов в выводе  short\* nrules; // номера правил грамматики  short\* nrulechains; // номера цепочек правил грамматики  Deducation() { size = 0; nrules = 0; nrulechains = 0; };  } deducation;  bool savededucation(); // сохранить дерево вывода  }; |

Листинг В.1 — Структура магазинного конечного автомата

# ПРИЛОЖЕНИЕ Г

|  |
| --- |
| Шаг : Правило Входная лента Стек  0 : S->tfi(F){NoR;};S tfi(ti){mti;i=ixi;oi;};p{ S$  0 : SAVESTATE: 1  0 : tfi(ti){mti;i=ixi;oi;};p{ tfi(F){NoR;};S$  1 : fi(ti){mti;i=ixi;oi;};p{m fi(F){NoR;};S$  2 : i(ti){mti;i=ixi;oi;};p{mt i(F){NoR;};S$  3 : (ti){mti;i=ixi;oi;};p{mti (F){NoR;};S$  4 : ti){mti;i=ixi;oi;};p{mti; F){NoR;};S$  5 : F->ti ti){mti;i=ixi;oi;};p{mti; F){NoR;};S$  5 : SAVESTATE: 2  5 : ti){mti;i=ixi;oi;};p{mti; ti){NoR;};S$  6 : i){mti;i=ixi;oi;};p{mti;i i){NoR;};S$  7 : ){mti;i=ixi;oi;};p{mti;i= ){NoR;};S$  8 : {mti;i=ixi;oi;};p{mti;i=l {NoR;};S$  9 : mti;i=ixi;oi;};p{mti;i=l; NoR;};S$  10 : N->mti; mti;i=ixi;oi;};p{mti;i=l; NoR;};S$  10 : SAVESTATE: 3  10 : mti;i=ixi;oi;};p{mti;i=l; mti;oR;};S$  11 : ti;i=ixi;oi;};p{mti;i=l;c ti;oR;};S$  12 : i;i=ixi;oi;};p{mti;i=l;c( i;oR;};S$  13 : ;i=ixi;oi;};p{mti;i=l;c(i ;oR;};S$  14 : i=ixi;oi;};p{mti;i=l;c(i) oR;};S$  15 : TS\_NOK/NS\_NORULECHAIN  15 : RESTATE  15 : mti;i=ixi;oi;};p{mti;i=l; NoR;};S$  16 : N->mti;N mti;i=ixi;oi;};p{mti;i=l; NoR;};S$  16 : SAVESTATE: 3  16 : mti;i=ixi;oi;};p{mti;i=l; mti;NoR;};S$  17 : ti;i=ixi;oi;};p{mti;i=l;c ti;NoR;};S$  18 : i;i=ixi;oi;};p{mti;i=l;c( i;NoR;};S$  19 : ;i=ixi;oi;};p{mti;i=l;c(i ;NoR;};S$  20 : i=ixi;oi;};p{mti;i=l;c(i) NoR;};S$  21 : N->i=E; i=ixi;oi;};p{mti;i=l;c(i) NoR;};S$  21 : SAVESTATE: 4  21 : i=ixi;oi;};p{mti;i=l;c(i) i=E;oR;};S$  22 : =ixi;oi;};p{mti;i=l;c(i)[ =E;oR;};S$  23 : ixi;oi;};p{mti;i=l;c(i)[w E;oR;};S$  24 : E->i ixi;oi;};p{mti;i=l;c(i)[w E;oR;};S$  24 : SAVESTATE: 5  24 : ixi;oi;};p{mti;i=l;c(i)[w i;oR;};S$  25 : xi;oi;};p{mti;i=l;c(i)[w( ;oR;};S$  26 : TS\_NOK/NS\_NORULECHAIN  26 : RESTATE  26 : ixi;oi;};p{mti;i=l;c(i)[w E;oR;};S$  27 : E->i() ixi;oi;};p{mti;i=l;c(i)[w E;oR;};S$  27 : SAVESTATE: 5  27 : ixi;oi;};p{mti;i=l;c(i)[w i();oR;};S$  28 : xi;oi;};p{mti;i=l;c(i)[w( ();oR;};S$  29 : TS\_NOK/NS\_NORULECHAIN  29 : RESTATE  29 : ixi;oi;};p{mti;i=l;c(i)[w E;oR;};S$  30 : E->i(W) ixi;oi;};p{mti;i=l;c(i)[w E;oR;};S$  30 : SAVESTATE: 5  30 : ixi;oi;};p{mti;i=l;c(i)[w i(W);oR;};S$ |

Листинг Г.1 — Работа синтаксического анализатора (фрагмент)

|  |
| --- |
| 0 : S->tfi(F){NoR;};S  4 : F->ti  8 : N->mti;N  12 : N->i=E;  14 : E->iM  15 : M->xE  16 : E->i  19 : R->i  23 : S->p{N};  25 : N->mti;N  29 : N->i=E;N  31 : E->l  33 : N->c(I)[N];N  35 : I->i  38 : N->w(R);  40 : R->l  45 : N->mti;N  49 : N->i=E;N  51 : E->lM  52 : M->xE  53 : E->l  55 : N->mti;N  59 : N->i=E;N  61 : E->lM  62 : M->xE  63 : E->l  65 : N->mti;N  69 : N->i=E;N  71 : E->(E)M  72 : E->i(W)M  74 : W->i  76 : M->xE  77 : E->l  79 : M->xE  80 : E->(E)  81 : E->lM  82 : M->xE  83 : E->l  86 : N->w(R);N  88 : R->i  91 : N->mti;N  95 : N->mti;N  99 : N->i=E;N  101 : E->l  103 : N->i=E;N  105 : E->l  107 : N->mti;N  111 : N->i=E;N  113 : E->i(W)  115 : W->i,W  117 : W->i  120 : N->w(R);N  122 : R->i  125 : N->mti;N  129 : N->i=E;N  131 : E->l  133 : N->c(I)[N];N  135 : I->i  138 : N->mti;N  142 : N->i=E;N  144 : E->i(W)  146 : W->i |

Листинг Г.2 — Результат работы синтаксического анализатора (фрагмент)

# ПРИЛОЖЕНИЕ Д

|  |
| --- |
| bool PN::PolishNotation(int n, LT::LexTable& lextable, IT::IdTable& idtable) {  std::queue<LT::Entry> current;  std::stack<LT::Entry> stack;  int i = n;  int priority;  int parCount = 0;  int indID = -1;  while (lextable.table[i].lexema != ';') {  priority = getP(lextable.table[i]);  if (lextable.table[i].lexema == 'i' &&  lextable.table[i + 1].lexema == '(') { // попали на вызов ф-ии  indID = lextable.table[i].indID;  i++;  priority = getP(lextable.table[i]);  while (priority < 2) {  if (priority == 0) {  current.push(lextable.table[i]);  parCount++;  }  else if (priority == 1)  stack.push(lextable.table[i]);  else if (priority == -2);  else if (priority == -1) {  while (getP(stack.top()) != 1) {  // выталкиваем стек пока не встретим открывашку  current.push(stack.top());  stack.pop();  }  stack.pop();  current.push({ '@', lextable.table[i].sn, lextable.table[i].idxLT, indID });  i++;  priority = getP(lextable.table[i]);  break;  }  i++;  priority = getP(lextable.table[i]);  }  }  if (priority == 0)  current.push(lextable.table[i]);  else if (priority == 1)  stack.push(lextable.table[i]);  else if (priority == 2 || priority == 3 ||  priority == 4 || priority == 5 || priority == 6) {  while (!stack.empty()) {  if (getP(stack.top()) >= priority) {  current.push(stack.top());  stack.pop();  }  else  break;  }  stack.push(lextable.table[i]);  }  else if (priority == -1) {  while (getP(stack.top()) != 1) {  // выталкиваем стек пока не встретим открывашку  current.push(stack.top());  stack.pop();  }  stack.pop();  }  else if (priority == -3)  break;  i++;  }  while (!stack.empty()) {  current.push(stack.top());  stack.pop();  }  current.push(lextable.table[i]);  //обратная польская запись построена  int posLast = i; // поз. последнего символа выр-я до польской записи  i = n;  for (i; i <= posLast; i++) {  lextable.table[i] = { '#' , -1 , i , -1 };  while (!current.empty()) {  lextable.table[i] = current.front();  lextable.table[i].idxLT = i;  current.pop();  i++;  }  }  return true;  }  int PN::getP(LT::Entry table) {  char token = table.lexema;  if (token == 'x')  token = table.operatorValue;  if (token == '~')  return 6;  else if (token == '\*' || token == '/')  return 5;  else if (token == '+' || token == '-')  return 4;  else if (token == '&')  return 3;  else if (token == '|')  return 2;  else if (token == '(')  return 1;  else if (token == ')')  return -1;  else if (token == ',')  return -2;  else if (token == ';')  return -3;  else return 0;  } |

Листинг Д.1 — Алгоритм преобразования выражений к польской записи

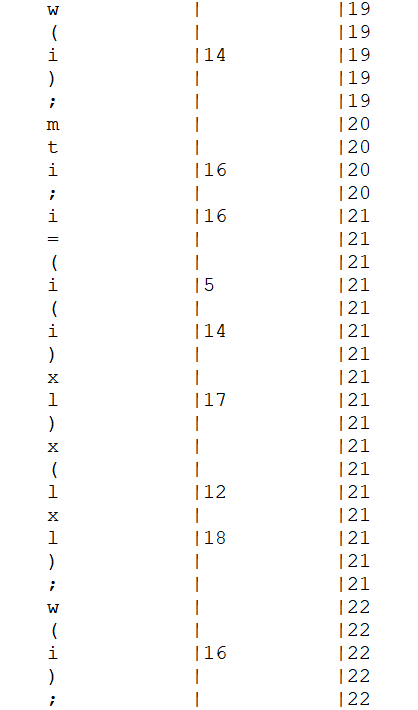
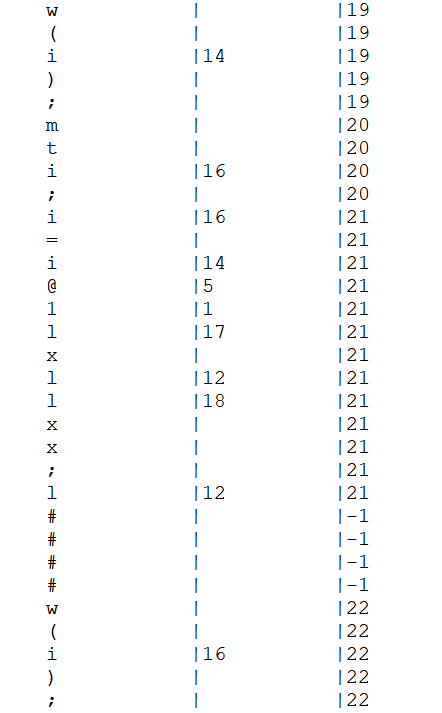
****

Рисунок Д.1 — Соответствие таблицы лексем и её аналога в польской нотации

# ПРИЛОЖЕНИЕ Е

|  |
| --- |
| .586P  .MODEL FLAT, STDCALL  includelib kernel32.lib  includelib libucrt.lib  includelib ..\Debug\Library.lib  EXTRN comp: PROC  EXTRN concat: PROC  EXTRN copy: PROC  EXTRN len: PROC  EXTRN ConsoleInt: PROC  EXTRN ConsoleStr: PROC  EXTRN ConsoleBool: PROC  EXTRN consolpause: proc  ExitProcess PROTO : DWORD  .STACK 4096  .CONST  L1 dd 1  L2 db "Hello!" , 0  L3 dd 15  L4 dd 5  L5 dd 20  L6 dd 50  L7 dd 4  L8 dd 16  L9 db "string1" , 0  L10 db "string2" , 0  L11 dd 7  L12 dd 0  .DATA  buffer BYTE 256 dup(0)  sum\_func DD 0  t\_Program DD 0  e\_Program DD 0  g\_Program DD 0  x\_Program DD 0  str1\_Program DD ?  str2\_Program DD ?  str3\_Program DD ?  y\_Program DD 0  s\_Program DD 0  .CODE  func PROC abc\_func : DWORD  push abc\_func  push abc\_func  pop EAX  pop EBX  add EAX, EBX  push EAX  pop sum\_func  mov eax, sum\_func  ret  func ENDP  Program PROC  START:  push L1  pop t\_Program  mov EAX,t\_Program  mov EBX, 1  sub EAX, EBX  je true1  jmp exit1  true1:  mov EAX, offset L2  push EAX  call ConsoleStr  exit1:  push L3  push L4  pop EAX  pop EBX  OR EAX, EBX  push EAX  pop e\_Program  push L5  push L6  pop EBX  pop EAX  sub EAX, EBX  push EAX  pop g\_Program  push g\_Program  call func  push EAX  push L7  pop EAX  pop EBX  add EAX, EBX  push EAX  push L3  push L8  pop EAX  pop EBX  AND EAX, EBX  push EAX  pop EBX  pop EAX  sub EAX, EBX  push EAX  pop x\_Program  mov EAX, x\_Program  push EAX  call ConsoleInt  push offset L9  pop str1\_Program  push offset L10  pop str2\_Program  push str1\_Program  push str2\_Program  push offset buffer  call concat  push EAX  pop str3\_Program  mov EAX, str3\_Program  push EAX  call ConsoleStr  push L1  pop y\_Program  mov EAX,y\_Program  mov EBX, 1  sub EAX, EBX  je true2  jmp exit2  true2:  push str1\_Program  call len  push EAX  pop s\_Program  mov EAX,s\_Program  mov EBX, L11  sub EAX, EBX  je true3  jmp exit3  true3:  mov EAX, s\_Program  push EAX  call ConsoleInt  exit3:  mov EAX, str2\_Program  push EAX  call ConsoleStr  exit2:  mov eax, L12  ret  push 0  call consolpause  call ExitProcess  Program ENDP  end Program |

Листинг Е.1 — Результат генерации ассемблерного кода