МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий
Кафедра программной инженерии
Специальность 6-05-0612-01 Программная инженерия

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:

«Разработка компилятора MDA-2024»

Выполнил студент Мамонько Денис Александрович
(Ф.И.О.)

Руководитель проекта преп.-ст. Некрасова А.П.
(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 преп.-ст. Некрасова А.П.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер преп.-ст. Некрасова А.П.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2024

Содержание

Введение

Целью данного курсового проекта является разработка компилятора для языка программирования MDA-2024.

Разработка компилятора выполняется на языке программирования C++. Код языка MDA-2024 будет генерироваться в язык ассемблера.

Компилятор состоит из следующих частей:

1. Лексический анализатор.
2. Синтаксический анализатор.
3. Семантический анализатор.
4. Генератор кода.

Для разработки компилятора необходимо выполнить следующие задачи:

1. Разработать спецификацию языка.
2. Разработать структуру транслятора.
3. Разработать лексический, синтаксический и семантический анализаторы.
4. Разработать алгоритм преобразования выражений.
5. Разработать алгоритм генерации кода на язык ассемблера.
6. Провести тестирование транслятора.

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

1 Спецификация языка программирования

1.1 Характеристика языка программирования

Язык программирования MDA-2024 является процедурным, универсальным, строго типизированным, не объектно-ориентированным, компилируемым.

1.2 Определение алфавита языка программирования

В алфавит языка программирования MDA-2024 входят символы латиницы ([a-z], [A-Z]), символы операций (+ - * / % > < ! & ~ @ { }) и сепараторов ([] () , ; : # пробел). Таблица входных символов представлена в пункте 3.2.

1.3 Применяемые сепараторы

Символы-сепараторы – это программные конструкции, служащие для разделения блоков кода. Они представлены в таблице 1.1.

Таблица 1.1 – Символы-сепараторы языка MDA-2024

Сепаратор	Название	Область применения
[…]	Квадратные скобки	Заключение программного блока
(…)	Круглые скобки	Блок фактических или формальных параметров функции, а также приоритет операций
,	Запятая	Разделитель параметров функций
“…”	Двойные кавычки	Строковый литерал
‘…’	Одинарные кавычки	Символьный литерал
#	Решётка	Символ, отделяющий условные конструкции/циклы
;	Точка с запятой	Разделитель программных конструкций
}{	Фигурные скобки	Операторы сдвигов
=	Знак «равно»	Оператор присваивания
+ - * / %	Знаки «плюс», «минус», «астерикс», «косая черта», «процент»	Арифметические операции
& ! ~ @ <>	Знаки «амперсанд», «восклицательный знак», «тильда», «собака», «меньше», «больше»	Операции сравнения

1.4 Применяемые кодировки

Для написания программ язык MDA-2024 использует кодировку ASCII, представленную на рисунке 1.1.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	00 □	01 □	02 □	03 □	04 □	05 □	06 □	07 □	08 □	09 □	0A □	0B □	0C □	0D □	0E □	0F □
10	10 □	11 □	12 □	13 □	14 □	15 □	16 □	17 □	18 □	19 □	1A □	1B □	1C □	1D □	1E □	1F □
20	20	21 !	22 “	23 #	24 $	25 %	26 &	27 (	28)	29 '	2A *	2B +	2C ,	2D -	2E .	2F /
30	30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7	38 8	39 9	3A :	3B ;	3C <	3D =	3E >	3F ?
40	40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G	48 H	49 I	4A J	4B K	4C L	4D M	4E N	4F O
50	50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W	58 X	59 Y	5A Z	5B [	5C \	5D]	5E ^	5F _
60	60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g	68 h	69 i	6A j	6B k	6C l	6D m	6E o	6F p
70	70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w	78 x	79 y	7A z	7B {	7C |	7D }	7E ~	7F □

Рисунок 1.1 – Таблица символов кодировки ASCII

1.5 Типы данных

Тип данных – это классификация данных, которая определяет, какие значения могут быть использованы, какие операции могут быть выполнены над этими значениями и как эти значения хранятся в памяти компьютера. В качестве основных типов данных, в языке программирования MDA-2024 реализованы 2 типа данных: целочисленный и символьный. Дополнительно был реализован строковый тип данных. Описание типов данных представлено в таблице 1.2.

Таблица 1.2 – Типы данных языка MDA-2024

Тип данных	Описание типа данных
Целочисленный, тип данных number	Представляет собой тип данных, предназначенный для хранения целых чисел без дробной части. Может хранить как положительные, так и отрицательные числа. Является одним из основных типов и используется для выполнения математических операций. Занимает 4 байта памяти. Диапазон значений: от -2,147,483,647 до 2,147,483,647. Инициализация по умолчанию – 0.
Символьный, тип данных char	Представляет собой тип данных, предназначенный для хранения одиночных символов, таких как буквы и цифры. Занимает 1 байт памяти. Диапазон значений: от 0 до 255. Инициализация по умолчанию – «\0» (символ конца строки).

Таблица 1.2 (Продолжение)

Тип данных	Описание типа данных
Строковый, тип данных line	Представляет собой тип данных, предназначенный для хранения последовательности символов. Может содержать буквы, цифры, пробелы и другие символы. Инициализация по умолчанию – "" (пустая строка).

1.6 Преобразование типов данных

Так как язык строго типизированный, то преобразование типов данных не поддерживается.

1.7 Идентификаторы

Общее количество идентификаторов ограничено максимальным размером таблицы идентификаторов. Идентификаторы должны содержать только символы нижнего регистра латинского алфавита. Максимальная длина идентификатора равна 16 символам. Идентификаторы, объявленные внутри функционального блока, получают префикс, идентичный имени функции, внутри которой они объявлены. Префикс занимает 16 дополнительных символов. В случае превышения заданной длины, идентификаторы усекаются до длины, равной 32 символа (16 символов на имя идентификатора, 16 символов на префикс). Данные правила действуют для всех типов идентификаторов. Зарезервированные идентификаторы не предусмотрены. Идентификаторы не должны совпадать с ключевыми словами. Типы идентификаторов: имя переменной, имя функции, параметр функции. Имена идентификаторов-функций не должны совпадать с именами команд ассемблера (это не касается имён идентификаторов-переменных).

1.8 Литералы

С помощью литералов осуществляется инициализация переменных. В языке программирования MDA-2024 присутствуют следующие типы литералов: целочисленный (десятичное и шестнадцатеричное представления), символьный, строковый. Описание литералов представлены в таблице 1.3.

Таблица 1.3 – Описание литералов языка MDA-2024

Тип литерала	Описание литерала
Целочисленный литерал в десятичном представлении	Целочисленный литерал может быть представлен в виде десятичного знакового числа. Литерал может быть только rvalue.

Таблица 1.3 (Продолжение)

Тип литерала	Описание литерала
Целочисленный литерал в шестнадцатеричном представлении	Целочисленный литерал может быть представлен в виде знакового шестнадцатеричного числа, в котором первые символы «0x» или «-0x» являются обязательными префиксами, после которых идет последовательность цифр от 0 до 9 и латинских букв от a до f с учетом регистра. Данный литерал также является только rvalue.
Строковый литерал	Строковый литерал состоит из последовательности символов латинского алфавита и цифр, заключенных в двойные кавычки. Данный литерал является rvalue.
Символьный литерал	Символьный литерал состоит из одиночного символа латинского алфавита или цифры от 0 до 9. Литерал является rvalue.

1.9 Объявление данных

Для объявления переменной используется ключевое слово var, после которого указывается тип данных и имя идентификатора. Допускается инициализация при объявлении. Для объявления функций используется ключевое слово function, перед которым указывается тип функции (если функция возвращает значение), или ключевое слово proc, если функция ничего не возвращает, а после – имя функции либо процедуры. Далее обязателен список параметров и тело функции.

1.10 Инициализация данных

Объектами-инициализаторами могут быть только идентификаторы или литералы. При объявлении предусмотрены значения по умолчанию: значение 0 для типа number и строка длины 0 (“”) для типа line. Также возможна инициализация непосредственно при объявлении переменной.

1.11 Инструкции языка

Инструкции языка MDA-2024 отображены в таблице 1.4.

Таблица 1.4 – Инструкции языка программирования MDA-2024

Инструкция	Запись на языке MDA-2024
Объявление переменной	var <тип данных> <идентификатор>;

Таблица 1.4 (Продолжение)

Инструкция	Запись на языке MDA-2024
Объявление функции	<тип данных> function <идентификатор> (<тип данных> <идентификатор>, …) [… return <идентификатор>;];
Объявление процедуры	proc function <идентификатор> (<тип данных> <идентификатор>, …) [… return;];
Присваивание	<идентификатор> = <выражение>; Выражением может быть литерал, идентификатор или вызов функции соответствующего типа.
Возврат из подпрограммы	return <выражение>; Выражением может быть литерал или идентификатор, для функции, которая не возвращает значение: return;
Вывод данных	print <идентификатор/литерал>;
Перевод строки	writeline;
Вызов функции или процедуры	<идентификатор функции> (<список параметров>); Список параметров может быть пустым.

1.12 Операции языка

В языке MDA-2024 предусмотрено несколько операций над данными. Все операции, с которыми можно работать представлены в таблице 1.5.

Таблица 1.5 – Операции языка программирования MDA-2024

Тип оператора	Оператор
Арифметические	1. + – сложение 2. - – вычитание 3. * – умножение 4. / – деление нацело 5. % – определение остатка от деления 6. = – присваивание
Строковые	1. = – присваивание
Сравнение	1. > – больше 2. < – меньше 3. & – проверка на равенство 4. !, @, ~ – проверка на неравенство
Сдвиговые	1. } – сдвиг вправо 2. { – сдвиг влево

1.13 Выражения и их вычисление

Вычисление выражений – одна из важнейших задач языков программирования. Всякое выражение составляется согласно следующим правилам:

1. Допускается использовать скобки для смены приоритета операций;
2. Выражение записывается в строку без переносов;
3. Использование двух подряд идущих операторов не допускается;
4. Допускается использовать в выражении вызов функции, вычисляющей возвращающей целочисленное значение.

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

1.14 Конструкции языка

Программа на языке MDA-2024 оформляется в виде функций пользователя и главной функции. При составлении функций рекомендуется выделять блоки и фрагменты и применять отступы для лучшей читаемости кода. Программные конструкции языка представлены в таблице 1.6.

Таблица 1.6 – Программные конструкции языка MDA-2024

Конструкция	Реализация
Главная функция	main [<программный блок>]
Функция	<тип данных> function <идентификатор>(<тип данных> <идентификатор>, …) [… return <идентификатор/литерал>;]
Процедура	proc function <идентификатор>(<тип данных><идентификатор>, …) [… return;]
Цикл	is: <идентификатор1> <оператор> <идентификатор2> # cycle […] # Цикл (операторы внутри блока cycle) выполняется, пока истинно условие “<идентификатор1> <оператор> <идентификатор2>”.

Таблица 1.6 (Продолжение)

Конструкция	Реализация
Условная конструкция	is: <идентификатор1> <оператор> <идентификатор2> # istrue […] isfalse […] # <идентификатор1>, <идентификатор2> – идентификаторы или литералы целочисленного типа (но не два литерала одновременно). <оператор> – один из операторов сравнения (> < & ! ~ @), устанавливающий отношение между двумя операндами и организующий условие данной конструкции. При истинности условия выполняется код внутри блока istrue, иначе – код внутри блока isfalse. Любой из блоков istrue, isfalse может отсутствовать, но не оба блока одновременно. При отсутствии одного из блоков, в зависимости от истинности или ложности условия программа может как выполнить один из заявленных блоков, так и передать управлении инструкции, следующей в коде за закрывающим условную конструкцию символом ‘#’.

1.15 Область видимости идентификаторов

Область видимости: сверху вниз (как и в C++). 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.

1.16 Семантические проверки

В языке программирования MDA-2024 есть следующие правила семантической проверки исходного текста языка, представленные в таблице 1.7.

Таблица 1.7 – Семантические проверки языка MDA-2024

Номер	Правило
1	Необъявленный идентификатор
2	Отсутствует точка входа main
3	Обнаружено несколько точек входа main
4	В объявлении не указан тип идентификатора
5	В объявлении отсутствует ключевое слово
6	Попытка переопределения идентификатора
7	Превышено максимальное количество параметров функции
8	Слишком много параметров в вызове

Таблица 1.7 (Продолжение)

Номер	Правило
9	Количество ожидаемых функцией и передаваемых параметров не совпадают
10	Несовпадение типов передаваемых параметров
11	Использование пустого строкового литерала недопустимо
12	Обнаружен символ \'\"\'. Возможно, не закрыт строковый литерал
13	Превышен размер строкового литерала
14	Недопустимый целочисленный литерал
15	Типы данных в выражении не совпадают
16	Тип функции и возвращаемого значения не совпадают
17	Недопустимое строковое выражение справа от знака \'=\'
18	Неверное условное выражение
19	Деление на ноль
20	Превышен размер символьного литерала

Назначение семантического анализа – проверка смысловой правильности конструкций языка программирования.

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 MDA-2024 присутствует стандартная библиотека, которая подключается автоматически при трансляции исходного кода в язык ассемблера. Содержимое библиотеки и описание основных функций представлено в таблице 1.8.

Таблица 1.8 – Основные функции стандартной библиотеки языка MDA-2024

Функция	Описание
line copystr (line str)	Строковая функция, предназначенная для копирования содержимого из строки str в другую.
number slength (line str)	Целочисленная функция, которая вычисляет и возвращает длину строки str.

Стандартная библиотека написана на языке C++, подключается к транслированному коду на этапе генерации кода. Вызовы стандартных функций доступны там же, где и вызов пользовательских функций. Также в стандартной библиотеке реализованы дополнительные функции для работы с числами и

строками и еще функции для манипулирования выводом, недоступные конечному пользователю. Для вывода предусмотрен оператор write. Эти функции представлены в таблицах 1.9 и 1.10 соответственно.

Таблица 1.9 – Дополнительные функции стандартной библиотеки языка MDA-2024

Функция	Описание
line concat (line str1, line str2)	Строковая функция, выполняющая объединение строк str1 и str2 в указанном порядке.
number atoii (line str)	Целочисленная функция, преобразующая строку в число.
number compare (line str1, line str2)	Целочисленная функция, которая принимает два строковых параметра, а затем сравнивает их. Если строки str1 и str2 равны, то функция вернет 1, если str1 меньше str2, то функция вернет 0, если str1 больше str2, то функция вернет 2.
number pow (number a, number b)	Целочисленная функция, которая принимает на вход два целочисленных параметра, возводит число a в степень b и возвращает результат.
number rnd (number a, number b)	Целочисленная функция, которая принимает на вход два целочисленных параметра и выводит случайное число в диапазоне от a до b.

Таблица 1.10 – Функции манипулирования выводом в стандартной библиотеке

Функция на языке C++	Описание
void outlich (int value)	Функция для вывода в стандартный поток значения целочисленного идентификатора/литерала.
void outrad(char* line)	Функция для вывода в стандартный поток значения строкового идентификатора/литерала

1.19 Ввод и вывод данных

Вывод данных осуществляется с помощью оператора print. Допускается использование оператора print с литералами и идентификаторами.

Функции, управляющие выводом данных, реализованы на языке C++ и вызываются из транслированного кода, конечному пользователю недоступны. Пользовательская команда print в транслированном коде будут заменена вызовом нужных библиотечных функций. Библиотека, содержащая нужные процедуры, подключается на этапе генерации кода.

1.20 Точка входа

В языке MDA-2024 каждая программа должна содержать главную функцию (точку входа) main. Данная функция может быть определена в программе только один раз и не может встречаться более одного раза или отсутствовать вообще. В случае нарушения данных условий будет зарегистрирована ошибка.

1.21 Препроцессор

В языке программирования MDA-2024 препроцессор отсутствует.

1.22 Соглашение о вызовах

В языке MDA-2024 вызов функций происходит по соглашению о вызовах stdcall. Особенности stdcall:

1. Все параметры функции передаются через стек;
2. Память высвобождает вызываемый код;
3. Занесение в стек параметров идёт справа налево.

1.23 Объектный код

Код на языке программирования MDA-2024 транслируется в исходный код на языке ассемблера.

1.24 Классификация сообщений транслятора

В случае возникновении ошибки в коде программы, в файл протокола будет выведено сообщение об ошибке с указанием места встречи этой ошибки. Классификация ошибок представлена в таблице 1.11.

Таблица 1.11 – Классификация ошибок

Номера ошибок	Характеристика
0 – 99	Системные ошибки
100 – 103	Ошибки входных параметров
200 – 204	Лексические ошибки
300 – 319	Семантические ошибки
600 – 615	Синтаксические ошибки

Обрабатываются ошибки на всех этапах обработки исходного кода, то есть во время прохождения различных этапов анализа.

1.25 Контрольный пример

Контрольный пример показывает работу всех функций и показывает особенности языка MDA-2024. Исходный код контрольного примера представлен в приложении А.

2 Структура транслятора

2.1 Компоненты транслятора, их назначение и принципы взаимодействия

В языке MDA-2024 исходный код транслируется в язык Assembler. Транслятор языка разделён на отдельные части, которые взаимодействуют между собой и выполняют отведённые им функции, которые представлены в пункте 2.1. Для того чтобы получить ассемблерный код, используется выходные данные работы лексического анализатора, а именно таблица лексем и таблица идентификаторов. Для указания выходных файлов используются входные параметры транслятора, которые описаны в таблице 2.1. Структура транслятора языка YSA-2024 приведена на рисунке 2.1.

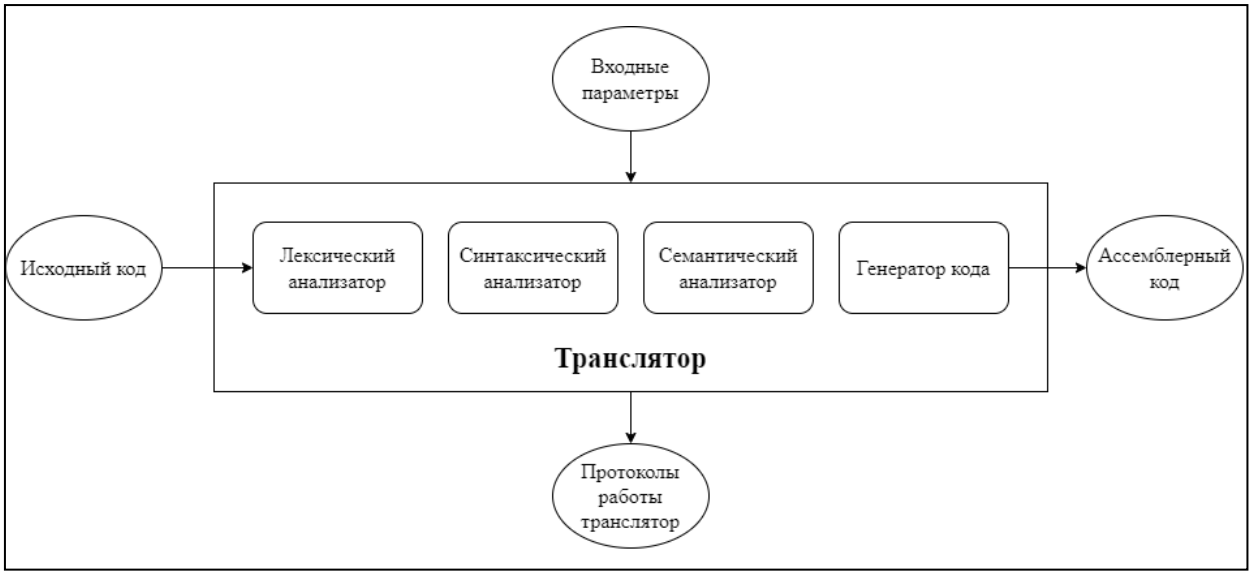

Рисунок 2.1 – Структура транслятора языка MDA-2024

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, парсера).

Цели лексического анализатора:

- убрать все лишние пробелы;

- выполнить распознавание лексем;
- построить таблицу лексем и таблицу идентификаторов;
- при неуспешном распознавании или обнаружении некоторых ошибок 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 анализатор – часть транслятора, выполняющая семантический анализ, то есть проверку исходного кода на наличие ошибок, которые невозможно отследить при помощи регулярной и контекстно-свободной грамматики. Входными данными являются таблица лексем и идентификаторов.

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

2.2 Перечень параметров транслятора

Для формирования файлов с результатами работы лексического, синтаксического и семантического анализаторов используются входные параметры транслятора, которые приведены в таблице 2.1.

Таблица 2.1 – Входные параметры транслятора языка MDA-2024

Входной параметр	Описание параметра	Значение по умолчанию
-in:<путь к in-файлу>	Файл с исходным кодом на языке MDA-2024, имеющий расширение .txt	Не предусмотрено
-log:<путь к log-файлу>	Файл журнала для вывода протоколов работы программы.	Значение по умолчанию: <имя in-файла>.log

2.3 Протоколы, формируемые транслятором

В ходе работы программы формируются протоколы работы лексического, синтаксического и семантического анализаторов, которые содержат в себе перечень протоколов работы. В таблице 2.2 приведены протоколы, формируемые транслятором и их содержимое.

Таблица 2.2 – Протоколы, формируемые транслятором языка MDA-2024

Формируемый протокол	Описание выходного протокола
Файл журнала, заданный параметром "-log:"	Файл с протоколом работы транслятора языка программирования MDA-2024.
Выходной файл, с расширением ".asm"	Результат работы программы – файл, содержащий исходный код на языке ассемблера.

Протокол работы содержит таблицу лексем и таблицу идентификаторов, протокол работы синтаксического анализатора и дерево разбора, полученные на этапе лексического и синтаксического анализа, а также результат работы алгоритма преобразования выражений к польской записи.

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 исходный код входного языка. Лексический анализатор выделяет в этой последовательности простейшие конструкции языка. Лексический анализатор производит предварительный разбор текста, преобразующий единый массив текстовых символов в массив токенов.

Функции лексического анализатора:

- удаление «пустых» символов и комментариев. Если «пустые» символы (пробелы, знаки табуляции и перехода на новую строку) и комментарии будут удалены лексическим анализатором, синтаксический анализатор никогда не столкнется с ними (альтернативный способ, состоящий в модификации грамматики для включения «пустых» символов и комментариев в синтаксис, достаточно сложен для реализации);
- распознавание идентификаторов и ключевых слов;
- распознавание констант;
- распознавание разделителей и знаков операций.

Исходный код программы представлен в приложении А, структура лексического анализатора представлена на рисунке 3.1.

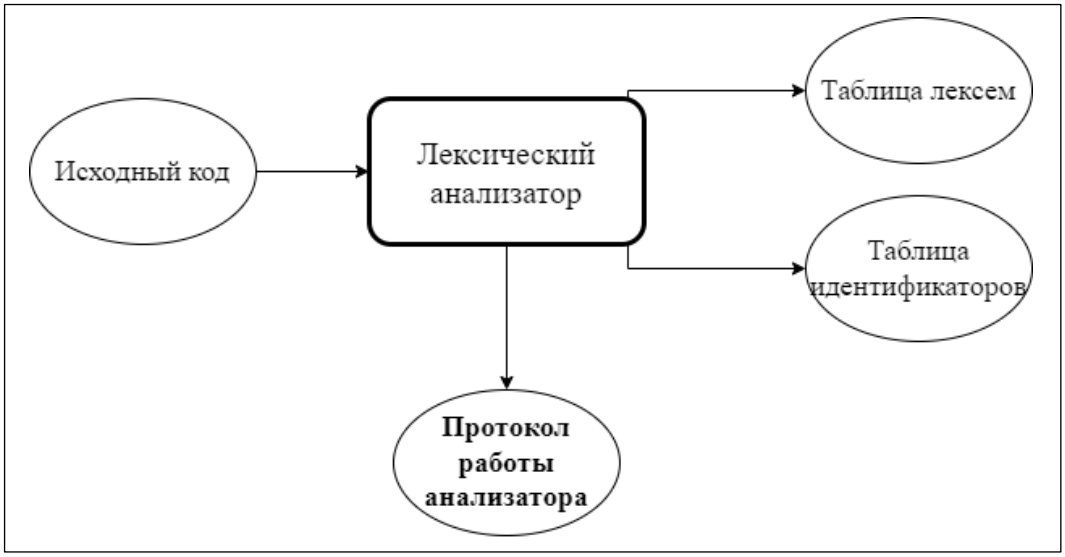

Рисунок 3.1 – Структура лексического анализатора

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.

3.2 Контроль входных символов

Для удобной работы с исходным кодом, при передаче его в лексический анализатор, все символы разделяются по категориям. Таблица входных символов представлена на рисунке 3.2, категории входных символов языка программирования MDA-2024 представлены в таблице 3.1.

```
#define IN_CODE_TABLE   {\
        IN::F, IN::F, IN::F, IN::F,  IN::F, IN::F, IN::F, IN::F,   IN::F, IN::P, IN::N, 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::P, IN::S, IN::Q, IN::S,  IN::T, IN::T, IN::S, IN::F,   IN::S, IN::S, IN::S, IN::S,  IN::S, 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::S,  IN::S, IN::S, 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::S,  IN::T, IN::S, IN::T, IN::T,\
        IN::F, IN::T, IN::T, IN::T,  IN::T, IN::T, IN::T, IN::T,   IN::T, IN::T, IN::T, IN::T,  IN::T, IN::T, IN::T, IN::T,\
        IN::T, IN::T, IN::T, IN::T,  IN::T, IN::T, IN::T, IN::T,   IN::T, IN::T, IN::T, IN::S,  IN::T, IN::T, 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::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 \
}
```

Рисунок 3.2 – Таблица контроля входных символов

Таблица 3.1 – Соответствие символов и их значений в таблице

Значение в таблице входных символов	Символы
Разрешенный	T
Запрещенный	F
Игнорируемый	I
Литерал	Q
Сепаратор	S
Перевод строки	N
Пробел, табуляция	P

3.3 Удаление избыточных символов

Избыточными символами являются символы табуляции и пробелы.

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 пробелов или знака табуляции является своего рода встречей символа-сепаратора.
3. В отличие от других символов-сепараторов не записываем в очередь лексем эти символы, т.е. игнорируем.

3.4 Перечень ключевых слов

Лексический анализатор преобразует исходный текст, заменяя лексические единицы лексемами для создания промежуточного представления исходной программы. Соответствие токенов и лексем приведено в таблице 3.2.

Таблица 3.2 – Соответствие токенов и сепараторов с лексемами

Токен	Лексема	Пояснение
number, line, char	t	Название типов данных языка.
Идентификатор	i	Длина идентификатора – 16 символов.
Литерал	l	Литерал любого доступного типа.
Шестнадцатеричный литерал	h	Целочисленный литерал в шестнадцатеричном представлении.
function	f	Объявление функции.
proc	p	Ключевое слово для процедур – функций, не возвращающих значение. Указывается перед словом function.
return	e	Выход из функции/процедуры.
main	m	Главная функция.
var	n	Объявление переменной
print	o	Вывод данных.
is:	?	Указывает на начало цикла/условного оператора.
istrue	r	Истинная ветвь условного оператора.
isfalse	w	Ложная ветвь условного оператора.
cycle	c	Указывает на начало тела цикла.
writeline	^	Оператор вывода символа перевода строки
#	#	Разделение конструкций в цикле/условном операторе.
;	;	Разделение выражений.
,	,	Разделение параметров функции.
[	[	Начало блока/тела функции.
]	]	Закрытие блока/тела функции.
}	}	Сдвиг вправо.
{	{	Сдвиг влево.
(	(	Передача параметров в функцию, приоритет операций.
)	)	Закрытие блока для передачи параметров, приоритет операций.
=	=	Знак присваивания.
+	+	Сложение.
-	-	Вычитание.
*	*	Умножение.
/	/	Деление нацело.
%	%	Остаток от деления.
>	>	Проверка на больше.
<	<	Проверка на меньше.
&	&	Проверка на равенство.
!	!	Проверка на неравенство.

Таблица 3.2 (Продолжение)

Токен	Лексема	Пояснение
~	~	Проверка на неравенство.
@	@	Проверка на неравенство.

Каждому выражению соответствует детерминированный конечный автомат, по которому происходит разбор данного выражения. На каждый автомат в массиве подаётся токен и с помощью регулярного выражения, соответствующего данному графу переходов, происходит разбор. В случае успешного разбора выражения оно записывается в таблицу лексем. Если выражение является идентификатором или литералом, информация также заносится в таблицу идентификаторов. Структура конечного автомата изображена на рисунке 3.3.

```
struct RELATION     // ребро: символ -> вершина графов переходов КА
{
    char  symbol;   // символ перехода
    short nnode;    // номер смежной вершины
    RELATION(
        char c,     // символ перехода
        short ns    // новое состояние
    );
};

struct NODE                 //вершина графа переходов
{
    short n_relation;       //количество инцидентных ребер
    RELATION *relations;    //инцидентные ребра
    NODE();                 //конструктор без параметров
    NODE(short n, RELATION rel, ...);  //количество инцидентных ребер, список ребер
};

struct FST   //недетерминированный конечный автомат
{
    char* string;               //цепочка(строка, завершается 0x00)
    short position;             //текущая позиция в цепочке
    short nstates;              //количество состояний автомата
    NODE* node;                 //граф переходов:[0]-начальное состояние, [nstate-1]-конечное
    short* rstates;             //возможные состояния автомата на данной позиции
    FST(short ns, NODE n, ...); // (массив)количество состояний автомата, список состояний(граф переходов)
    FST(char* s, FST& fst);     // количество состояний автомата, список состояний(граф переходов)
```

Рисунок 3.3 – Структура конечного автомата

Пример графа перехода конечного автомата изображен на рисунке 3.4.

```
#define GRAPH_VAR 4, \
    FST::NODE(1,FST::RELATION('v',1)),\
    FST::NODE(1,FST::RELATION('a',2)),\
    FST::NODE(1,FST::RELATION('r',3)),\
    FST::NODE()
```

Рисунок 3.4 – Пример реализации графа КА для токена var

Пример реализации таблицы лексем представлен в приложении Б.

3.5 Основные структуры данных

Основными структурами данных лексического анализатора являются таблица лексем и таблица идентификаторов. Таблица лексем содержит номер лексемы,

лексему (lexema), полученную при разборе, номер строки в исходном коде (sn), и номер в таблице идентификаторов, если лексема является идентификаторов (idxTI). Код C++ со структурой таблицы лексем представлен на рисунке 3.5.

```
struct Entry
{
    char lexema;                    //лексема
    int sn;                         //номер строки в исходном тексте
    int idxTI;                      //индекс в ТИ

    Entry();
    Entry(char lexema, int snn, int idxti = NULLDX_TI);
};

struct LexTable                     //экземпляр таблицы лексем
{
    int maxsize;                    //ёмкость таблицы лексем
    int size;                       //текущий размер таблицы лексем
    Entry* table;                   //массив строк ТЛ
};
```

Рисунок 3.5 – Структура таблицы лексем

Таблица идентификаторов содержит имя идентификатора (id), номер в таблице лексем (idxfirstLE), тип данных (iddatatype), тип идентификатора (idtype) и значение (или параметры функций) (value). Код C++ со структурой таблицы идентификаторов представлен на рисунке 3.6.

```
struct Entry
{
    union
    {
        int vint;                   //значение integer
        struct
        {
            int len;                //количество символов
            char str[STR_MAXSIZE - 1];//символы
        } vstr;                     //значение строки
        struct
        {
            int count;              // количество параметров функции
            IDDATATYPE* types;      //типы параметров функции
        } params;
    } value;                      //значение идентификатора
    int         idxfirstLE;         //индекс в таблице лексем
    char        id[SCOPED_ID_MAXSIZE]; //идентификатор
    IDDATATYPE  iddatatype;         //тип данных
    IDTYPE      idtype;             //тип идентификатора

    Entry() { ... }
    Entry(char* id, int idxLT, IDDATATYPE datatype, IDTYPE idtype) { ... }
};
struct IdTable      //экземпляр таблицы идентификаторов
{
    int maxsize;    //ёмкость таблицы идентификаторов < TI_MAXSIZE
    int size;       //текущий размер таблицы идентификаторов < maxsize
    Entry* table;   //массив строк таблицы идентификаторов
};
```

Рисунок 3.6 – Структура таблицы идентификатора

3.6 Структура и перечень сообщений лексического анализатора

Для обработки ошибок лексический анализатор использует таблицу с сообщениями. Структура сообщений содержит информацию о номере сообщения, номер строки и позицию, где было вызвано сообщение в исходном коде, информацию об ошибке. Перечень сообщений об ошибках лексического анализатора представлен на рисунке 3.7.

```
ERROR_ENTRY(200, "Лексическая ошибка: Недопустимый символ в исходном файле(-in)"),
ERROR_ENTRY(201, "Лексическая ошибка: Неизвестная последовательность символов"),
ERROR_ENTRY(202, "Лексическая ошибка: Превышен размер таблицы лексем"),
ERROR_ENTRY(203, "Лексическая ошибка: Превышен размер таблицы идентификаторов"),
ERROR_ENTRY(204, "Лексическая ошибка: Превышено число символов идентификатора"),
```

Рисунок 3.7 – Сообщения лексического анализатора

При возникновении сообщения, лексический анализатор выбрасывает исключение – работа программы останавливается.

3.7 Принцип обработки ошибок

Ошибки, возникающие в процессе трансляции программы, фиксируются в протокол, заданный входным параметрами. В случае возникновения ошибок происходит их протоколирование с номером ошибки и диагностическим сообщением.

3.8 Параметры лексического анализатора

Результаты работы лексического анализатора, а именно таблицы лексем и идентификаторов выводятся как в файл журнала, так и в командную строку.

3.9 Алгоритм лексического анализа

Описание алгоритма лексического анализа следующее:
1. Лексический анализатор проверяет входной поток символов программы на исходном языке на допустимость, удаляет лишние пробелы и добавляет сепаратор для вычисления номера строки для каждой лексемы;
2. Для выделенной части входного потока выполняется функция распознавания лексемы;
3. При успешном распознавании информация о выделенной лексеме заносится в таблицу лексем и таблицу идентификаторов, и алгоритм возвращается к первому этапу;
4. Формирует протокол работы;
5. При неуспешном распознавании выдается сообщение об ошибке.

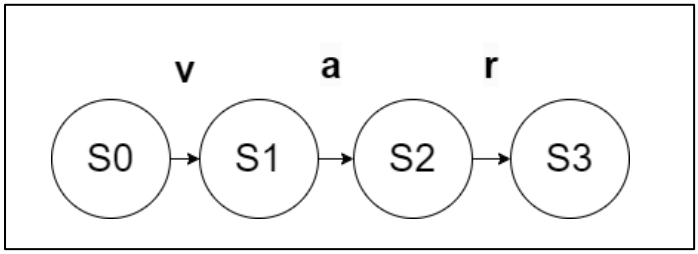

Рисунок 3.8 – Пример графа переходов для цепочки var

Распознавание цепочек основывается на работе конечных автоматов. Работу конечного автомата можно проиллюстрировать с помощью графа переходов. Пример

графа для цепочки «var» представлен на рисунке 3.8, где S0 – начальное, а S3 – конечное состояние автомата.

3.10 Контрольный пример

Результат работы лексического анализатора в виде таблицы лексем и идентификаторов, соответствующих контрольному примеру, представлен в приложении В.

4 Разработка синтаксического анализатора

4.1 Структура синтаксического анализатора

Синтаксический анализатор: часть компилятора, выполняющая синтаксический анализ, то есть исходный код проверяется на соответствие правилам грамматики. Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов. Выходной информацией– дерево разбора.

Описание структуры синтаксического анализатора языка представлено на рисунке 4.1.

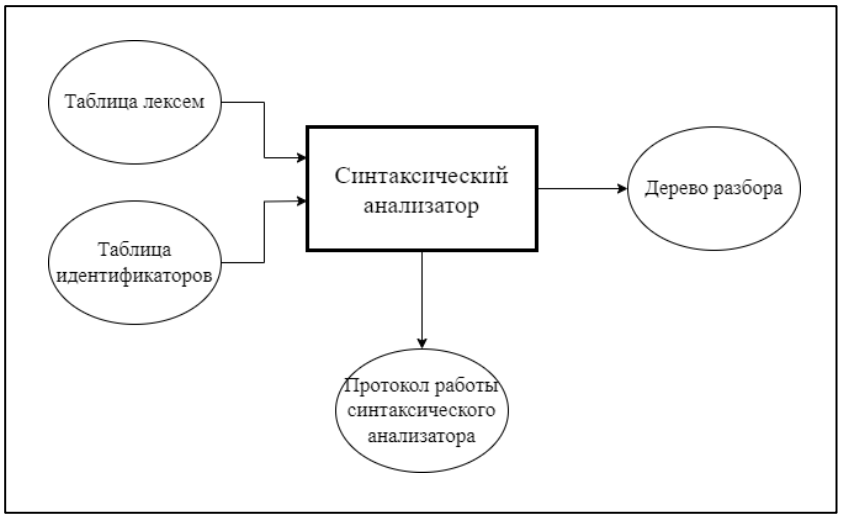

Рисунок 4.1 – Структура синтаксического анализатора

4.2 Контекстно-свободная грамматика, описывающая синтаксис языка

В синтаксическом анализаторе транслятора языка MDA-2024 используется контекстно-свободная грамматика $G = \langle T, N, P, S \rangle$, где

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

N – множество нетерминальных символов,

P – множество правил языка,

S – начальный символ грамматики, являющийся нетерминалом.

Эта грамматика имеет нормальную форму Грейбах, т.к. она не леворекурсивная (не содержит леворекурсивных правил)

Таблица 4.1, описывающая правила грамматики языка MDA-2024 представлена в приложении Г.

4.3 Построение конечного магазинного автомата

Конечный автомат с магазинной памятью представляет собой семерку $M = \langle Q, V, Z, \delta, q_0, z_0, F \rangle$. Подробное описание компонентов магазинного автомата представлено в таблице 4.2.

Таблица 4.2 – Описание компонентов магазинного автомата

Компонент	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 представляет из себя множества терминальных и нетерминальных символов, описание которых содержится в таблица 3.1 и 4.1.
Z	Алфавит специальных магазинных символов	Алфавит магазинных символов содержит стартовый символ и маркер дна стека (представляет из себя символ $)
δ	Функция переходов автомата	Функция представляет из себя множество правил грамматики, описанных в таблице 4.1.
q_0	Начальное состояние автомата	Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики
z_0	Начальное состояние магазина 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

На основании данной таблицы можно заключить, что магазинный автомат применяется для анализа и обработки языка MDA-2024 с использованием контекстно-свободной грамматики Автомат включает состояния, алфавит символов, функцию переходов, а также начальное и конечные состояния. Используя эти компоненты, автомат выполняет разбор и трансляцию программного кода, написанного на языке MDA-2024.

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 MDA-2024. Данные структуры представлены в приложении Д.

4.5 Описание алгоритма синтаксического разбора

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

4.6 Структура и перечень сообщений синтаксического анализатора

Перечень сообщений синтаксического анализатора, которые можно получить, представлен на рисунке 4.2.

```
ERROR_ENTRY(600, "Синтаксическая ошибка: Неверная структура программы"),
ERROR_ENTRY(601, "Синтаксическая ошибка: Не найден список параметров функции"),
ERROR_ENTRY(602, "Синтаксическая ошибка: Ошибка в теле функции"),
ERROR_ENTRY(603, "Синтаксическая ошибка: Ошибка в теле процедуры"),
ERROR_ENTRY(604, "Синтаксическая ошибка: Ошибка в списке параметров функции"),
ERROR_ENTRY(605, "Синтаксическая ошибка: Ошибка в вызове функции/выражении"),
ERROR_ENTRY(606, "Синтаксическая ошибка: Ошибка в списке фактических параметров функции"),
ERROR_ENTRY(607, "Синтаксическая ошибка: Ошибка при констуировании цикла/условного выражения"),
ERROR_ENTRY(608, "Синтаксическая ошибка: Ошибка в теле цикла/условного выражения"),
ERROR_ENTRY(609, "Синтаксическая ошибка: Ошибка в условии цикла/условного выражения"),
ERROR_ENTRY(610, "Синтаксическая ошибка: Неверный условный оператор"),
ERROR_ENTRY(611, "Синтаксическая ошибка: Неверный арифметический оператор"),
ERROR_ENTRY(612, "Синтаксическая ошибка: Неверное выражение. Ожидаются только идентификаторы/литералы"),
ERROR_ENTRY(613, "Синтаксическая ошибка: Ошибка в арифметическом выражении"),
ERROR_ENTRY(614, "Синтаксическая ошибка: Недопустимая синтаксическая конструкция"),
ERROR_ENTRY(615, "Синтаксическая ошибка: Недопустимая синтаксическая конструкция в теле цикла/условного выражения"),
```

Рисунок 4.2 – Сообщения синтаксического анализатора

4.7 Параметры синтаксического анализатора и режимы его работы

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

4.8 Принцип обработки ошибок

Синтаксический анализатор выполняет разбор исходной последовательности лексем до тех пор, пока не дойдёт до конца цепочки лексем или не найдёт ошибку.

Тогда анализ останавливается и выводится сообщение об ошибке (если она найдена). Если в процессе анализа находятся более трёх ошибок, то анализ останавливается.

4.9 Контрольный пример

Результаты работы лексического разбора, а именно дерево разбора и протокол работы автомата с магазинной памятью приведены в приложении Е.

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.

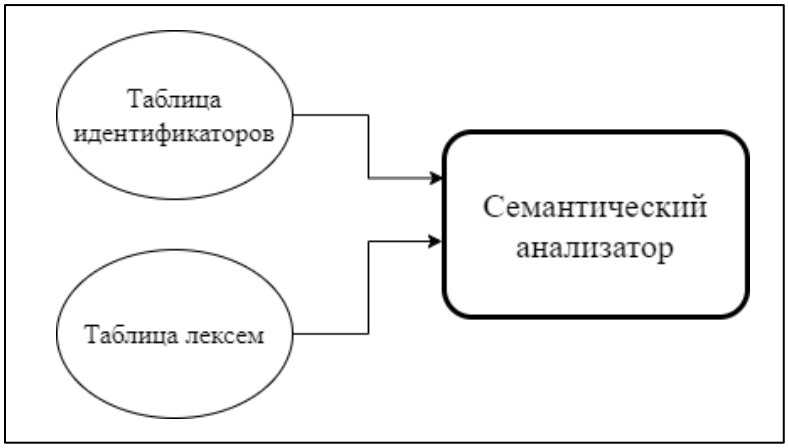

Рисунок 5.1 – Структура семантического анализатора

5.2 Функции семантического анализатора

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

5.3 Структура и перечень сообщений семантического анализатора

Сообщения, формируемые семантическим анализатором, представлены на рисунке 5.2.

```
ERROR_ENTRY(300, "Семантическая ошибка: Необъявленный идентификатор"),
ERROR_ENTRY(301, "Семантическая ошибка: Отсутствует точка входа main"),
ERROR_ENTRY(302, "Семантическая ошибка: Обнаружено несколько точек входа main"),
ERROR_ENTRY(303, "Семантическая ошибка: В объявлении не указан тип идентификатора"),
ERROR_ENTRY(304, "Семантическая ошибка: В объявлении отсутствует ключевое слово"),
ERROR_ENTRY(305, "Семантическая ошибка: Попытка переопределения идентификатора"),
ERROR_ENTRY(306, "Семантическая ошибка: Превышено максимальное количество параметров функции"),
ERROR_ENTRY(307, "Семантическая ошибка: Слишком много параметров в вызове"),
ERROR_ENTRY(308, "Семантическая ошибка: Кол-во ожидаемых функцией и передаваемых параметров не совпадают"),
ERROR_ENTRY(309, "Семантическая ошибка: Несовпадение типов передаваемых параметров"),
ERROR_ENTRY(310, "Семантическая ошибка: Использование пустого строкового литерала недопустимо"),
ERROR_ENTRY(311, "Семантическая ошибка: Обнаружен символ \'\"\'. Возможно, не закрыт строковый литерал"),
ERROR_ENTRY(312, "Семантическая ошибка: Превышен размер строкового литерала"),
ERROR_ENTRY(313, "Семантическая ошибка: Недопустимый целочисленный литерал"),
ERROR_ENTRY(314, "Семантическая ошибка: Типы данных в выражении не совпадают"),
ERROR_ENTRY(315, "Семантическая ошибка: Тип функции и возвращаемого значения не совпадают"),
ERROR_ENTRY(316, "Семантическая ошибка: Недопустимое строковое выражение справа от знака \'=\'"),
ERROR_ENTRY(317, "Семантическая ошибка: Неверное условное выражение"),
ERROR_ENTRY(318, "Семантическая ошибка: Деление на ноль"),
ERROR_ENTRY(319, "Семантическая ошибка: Превышен размер символьного литерала"),
```

Рисунок 5.2 – Перечень сообщений семантического анализатора

5.4 Принцип обработки ошибок

Ошибки, возникающие в процессе трансляции программы, фиксируются в протокол, заданный входным параметрами. В случае возникновения ошибок происходит их протоколирование с номером ошибки и диагностическим сообщением. Анализ останавливается после того, как будут найдены все ошибки.

5.5 Контрольный пример

Соответствие примеров некоторых ошибок в исходном коде и диагностических сообщений об ошибках приведено в таблице 5.1.

Таблица 5.1 – Примеры диагностики ошибок

Исходный код	Текст сообщения
main[] main[var number num = 2; print num;]	Ошибка 302: Семантическая ошибка: Обнаружено несколько точек входа main. Строка: 0.
number function min(number x, number x) [var number res; is: x > y # istrue [res = x;] isfalse [res = y;]# return res;]	Ошибка 305: Семантическая ошибка: Попытка переопределения идентификатора. Строка: 1.

main
[
var number f = -4;
var number s = 0;
var number finish;
finish = f / s;
print finish;
]
	Ошибка 318: Семантическая ошибка: Деление на нуль.
	Строка: 33.

6 Вычисление выражений

6.1 Выражения, допускаемые языком

В языке MDA-2024 допускаются вычисления выражений целочисленного типа данных с поддержкой вызова функций внутри выражений. Приоритет операций представлен в таблице 6.1.

Таблица 6.1 – Приоритеты операций

Операция	Значение приоритета
()	3
*	2
/	2
%	2
+	1
-	1
}	0
{	0

6.2 Польская запись и принцип её построения

Все выражения языка MDA-2024 преобразовываются к обратной польской записи.

Польская запись — это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок. Существует два типа польской записи: прямая и обратная, также известные как префиксная и постфиксная. Отличие их от классического, инфиксного способа заключается в том, что знаки операций пишутся не между, а, соответственно, до или после аргументов. Алгоритм построения польской записи:

- исходная строка: выражение;
- результирующая строка: польская запись;
- стек: пустой;
- исходная строка просматривается слева направо;
- операнды переносятся в результирующую строку;
- операция записывается в стек, если стек пуст;
- операция выталкивает все операции с большим или равным приоритетом в результирующую строку;
- отрывающая скобка помещается в стек;
- закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются.

6.3 Программная реализация обработки выражений

Программная реализация алгоритма преобразования выражений в обратную польскую запись основана на функции PolishNotation. Функция принимает как параметр таблицу лексем и таблицу идентификаторов и содержит цикл, в ходе

которого перебираются все лексемы исходного кода. Если последовательность лексем соответствует началу выражения, то проводится преобразование выражений к польской записи.

6.4 Контрольный пример

В приложении Ж приведено представление промежуточного кода, отображающее результаты преобразования выражений в польский формат.

7 Генерация кода

7.1 Структура генератора кода

В языке MDA-2024 генерация кода является заключительным этапом трансляции. Генератор принимает на вход таблицы лексем и идентификаторов, полученные в результате лексического анализа. В соответствии с таблицей лексем строится выходной файл на языке ассемблера, который будет являться результатом работы транслятора. В случае возникновения ошибок генерация кода не будет осуществляться. Структура генератора кода MDA-2024 представлена на рисунке 7.1.

Рисунок 7.1 – Структура генератора кода

7.2 Представление типов данных в оперативной памяти

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

Таблица 7.1 – Соответствия типов данных идентификаторов языка MDA-2024 и языка ассемблера

Тип идентификатора на языке MDA-2024	Тип идентификатора на языке ассемблера	Пояснение
number	sdword	Хранит целочисленный тип данных.
line	dword	Хранит указатель на начало строки. Строка должна завершаться нулевым символом.

Таблица 7.1(Продолжение)

Тип идентификатора на языке MDA-2024	Тип идентификатора на языке ассемблера	Пояснение
char	byte	Хранит символьный тип данных.

7.3 Статическая библиотека

В языке MDA-2024 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++. Объявление функций статической библиотеки генерируется автоматически в коде ассемблера. Объявление функций статической библиотеки генерируется автоматически. Функции статической библиотеки описаны в таблице 7.2.

Таблица 7.2 – Функции статической библиотеки

Функция	Назначение
void outrad(char* rad)	Вывод на консоль строки rad.
void outlich(int lich)	Вывод на консоль целочисленной переменной lich.
int slength(char* buffer, char* rad)	Вычисление длины строки rad.
char* concat(char* buffer, char* rad1, char* rad2)	Объединение строк rad1 и rad2.
int atoii(char* ptr)	Преобразование строки в число.
int compare(char* buffer, char* str1, char* str2)	Лексикографическое сравнение строк.
int poww(char* ptr, int num, int exponent)	Возведение числа num в степень exponent.
int rnd(char* ptr, int a, int b)	Генерация случайного числа в диапазоне от a до b.
char* copy(char* buffer, char*str)	Копирование из строки str в строку buffer.

7.4 Особенности алгоритма генерации кода

В языке MDA-2024 генерация кода строится на основе таблиц лексем и идентификаторов. Общая схема работы генератора кода представлена на рисунке 7.2.

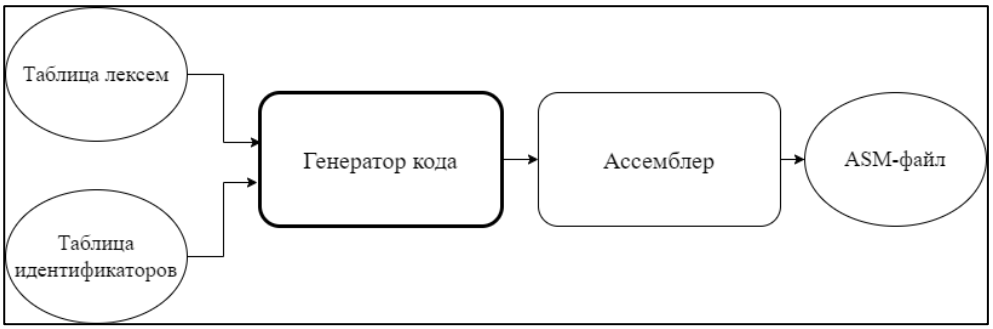

Рисунок 7.2 – Структура генератора кода

7.5 Входные параметры генератора кода

На вход генератору кода поступают таблицы лексем и идентификаторов исходного код программы на языке MDA-2024. Результаты 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 З.

8 Тестирование транслятора

8.1 Общие положения

Тестирование должно покрывать как можно больше сценариев использования языка и его конструкций. Все тесты были представлены для типичных ошибок пользователей при использовании языка. Когда компилятор обнаруживает ошибку, он записывает информацию о ней в протокол, содержащий номер ошибки и диагностическое сообщение, помогающее разработчику понять причину ошибки компиляции. Результаты тестирования записываются в файл .log.

8.2 Результаты тестирования

В таблице 8.1 приведены результаты тестов для разных этапов трансляции.

Таблица 8.1 – Тестирование лексического анализатора

Исходный код	Диагностическое сообщение
Проверка символов на допустимость	
main [print “NumЙber”;]	Ошибка 200: Лексическая ошибка: Недопустимый символ в исходном файле (-in). Строка: 3, позиция: 11
Лексический анализ	
main [var number sixasfasdasdasdadasd = 2;]	Ошибка 204: Лексическая ошибка: Превышено число символов идентификатора. Строка: 3.
main [var number six = 0x4as12a;]	Ошибка 201: Лексическая ошибка: Неизвестная последовательность символов Строка: 3.
Синтаксический анализ	
proc function stand() [var line str; print str; return str;]	Ошибка 614: Синтаксическая ошибка: Недопустимая синтаксическая конструкция. Строка: 5.

main
[
var number f = -4;
var number s = 2;
var number finish;
finish = f // s;
]
	Ошибка 613: Синтаксическая ошибка: Ошибка в арифметическом выражении.
	Ошибка 611: Синтаксическая ошибка: Неверный арифметический оператор.
	Строка: 6.

Таблица 8.1 (Продолжение)

Исходный код	Диагностическое сообщение
number function min(number x, number y) [var number res; is: x > y / istrue [res = x;] isfalse [res = y;]# return res;]	Ошибка 610: Синтаксическая ошибка: Неверный условный оператор. Строка: 4.
Семантический анализ	
main[] main[var number num = 2; print num;]	Ошибка 302: Семантическая ошибка: Обнаружено несколько точек входа main. Строка: 0.
number function min(number x, number x) [var number res; is: x > y # istrue [res = x;] isfalse [res = y;]# return res;]	Ошибка 305: Семантическая ошибка: Попытка переопределения идентификатора. Строка: 1.

main
[
var number f = -4;
var number s = 0;
var number finish;
finish = f / s;
print finish;
]
	Ошибка 318: Семантическая ошибка: Деление на нуль.
	Строка: 5.

main [var str; print str;]	Ошибка 304: Семантическая ошибка: В объявлении отсутствует ключевое слово. Строка: 3.

Таким образом данный раздел предоставляет набор тестов для проверки лексического, синтаксического и семантического анализаторов.

Заключение

В ходе выполнения курсовой работы был разработан транслятор и генератор кода для языка программирования MDA-2024 со всеми необходимыми компонентами. Таким образом, были выполнены основные задачи данной курсовой работы:

Сформулирована спецификация языка MDA-2024;

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

Окончательная версия языка MDA-2024 включает:

1. 3 типа данных;
2. Поддержка оператора вывода;
3. Возможность вызова 7 функций стандартной библиотеки;
4. Наличие 5 арифметических операторов для вычисления выражений, наличие 6 операторов сравнения для сравнения выражений, 2 оператора для сдвиговых операций;
5. Поддержка функций, операторов цикла и условия;
6. Структурированная и классифицированная система для обработки ошибок пользователя.

Проделанная работа позволила получить необходимое представление о структурах и процессах, использующихся при построении трансляторов, а также основные различия и преимущества тех или иных средств трансляции.

Список использованных источников

1. Вирт Н. Построение компиляторов/ Пер. с англ. Борисов Е. В., Чернышов Л. Н. – М.: ДМК Пресс, 2010. – 192 с.: ил.

2. Грис Д. Конструирование компиляторов для цифровых вычислительных машин.: Пер. с англ. — М.: Мир, 1975.

3. Костельцев А. В. Построение интерпретаторов и компиляторов. СПб: Наука и Техника, 2001. — 224 стр. с ил.

4. Пратт Т. Языки программирования: разработка и реализация. Пер. с англ. — М.: Мир, 1979.

5. Хантер Р. Проектирование и конструирование компиляторов/ Пер. с англ.: Предисл. В. М. Савинкова. — М.: Финансы и статистика, 1984. — 232 с., ил.

6. Хендрикс Д. Компилятор языка Си для микроЭВМ: Пер. с англ. — М.: радио и связь, 1989. — 240 с.: ил.

Приложение А

```
number function min(number x, number y)
[
        var number  res;
        is: x > y #
        istrue [res = x;]
        isfalse [res = y;]#
        return res;
]

proc function stand(line a, line b)
[
        var line str;
        str = concat(a,b);
       print "Result: ";
        print str;
        writeline;
        return;
]

main[
        print "Number at sixteen format: ";
        var number six = 0x4a;
        print six;
        writeline;
        print "----------------------";
        writeline;

        var line str;
        var number abc = 2;
        var number abcd = 4;

        var number f = -4;
        var number s = 2;
        var number finish;
        finish = f / s;
        print "result of division: ";
        print finish;
        writeline;
        print "----------------------";
        writeline;

        print "("; print abc; print ">="; print abcd; print ") :";
        is: abc ~ abcd #
```

```
istrue [str="True";]
isfalse [str="False";]#
print str;
writeline;
print "----------------------";
writeline;

print "Copited string: ";
var line stroka = "Denis";
print stroka;
writeline;
var line strochka;
strochka = copystr(stroka);
print "----------------------";
writeline;

var char mh = 'H';
var char mi = 'i';
var char miv = '!';
print mh;
print mi;
print miv;
writeline;
print "----------------------";
writeline;

var line dsa = "False";
var line ytr = "False";
print "Compare: ";
var number  asd;
asd=compare(dsa,ytr);
is: asd & 1 #
istrue [print "The lines are the same";]
isfalse [print "The lines are not the same";]#
writeline;
print "----------------------";
writeline;

print "Exponentiation of a number: ";
var number  p = 6;
p = poww(2,p);
print p;
writeline;
print "----------------------";
writeline;
```

```
print "Random number: ";
var number x;
x = rnd(1, 10);
print x;
writeline;
print "----------------------";
writeline;

var number  u = 5;
var number  v = 7;
var number  r;
print "Arithmetic expression:  ";
r=u+v*2-(4+4)/2 + min(5, 8);
print r;
writeline;
print "----------------------";
writeline;

var line c = "123";
var number e;
print "Converting a string to a number: ";
e = atoii(c);
print e;
writeline;
print "----------------------";
writeline;

var number k;
var line len="I love SE";
k = slength(len);
print "Line length: ";
print k;
writeline;
print "----------------------";
writeline;

var number numb = 54;
var number remainder;
remainder = numb % 3;
print "Remainder after division 54 by 3: ";
print remainder;
writeline;
print "----------------------";
writeline;
```

```
	var number  result;
	var number sdv=3;
	result = sdv{2;
	print "Number after shifted to the left : ";
	print result;
	writeline;
	print "----------------------";
	writeline;


	var line name ="Denis ";
	var line surname ="Mamonko";

	var number poi = 4;
	var number isi = 4;
	is: poi @ isi #
	istrue [stand(name,surname);]
	isfalse [print "Error";]#
	print "----------------------";
	writeline;

	var number  ab = 2;
	print "Cycle from 2 to 10: ";
	is: ab ! 10 #
	cycle [
		print ab;
		print " ";
		ab = ab + 2;
	]#
	print ab;
	writeline;
]
```

Листинг 1 – Исходный код на языке MDA-2024

Приложение Б

```
#define GRAPH_ISTRUE 7, \
        FST::NODE(1, FST::RELATION('i',1)),\
        FST::NODE(1, FST::RELATION('s',2)),\
        FST::NODE(1, FST::RELATION('t',3)),\
        FST::NODE(1, FST::RELATION('r',4)),\
        FST::NODE(1, FST::RELATION('u',5)),\
        FST::NODE(1, FST::RELATION('e',6)),\
        FST::NODE()


#define GRAPH_ISFALSE 8, \
        FST::NODE(1, FST::RELATION('i',1)),\
        FST::NODE(1, FST::RELATION('s',2)),\
        FST::NODE(1, FST::RELATION('f',3)),\
        FST::NODE(1, FST::RELATION('a',4)),\
        FST::NODE(1, FST::RELATION('l',5)),\
        FST::NODE(1, FST::RELATION('s',6)),\
        FST::NODE(1, FST::RELATION('e',7)),\
        FST::NODE()

#define GRAPH_FUNCTION 9, \
        FST::NODE(1, FST::RELATION('f', 1)),\
   FST::NODE(1, FST::RELATION('u', 2)),\
   FST::NODE(1, FST::RELATION('n', 3)),\
   FST::NODE(1, FST::RELATION('c', 4)),\
   FST::NODE(1, FST::RELATION('t', 5)),\
   FST::NODE(1, FST::RELATION('i', 6)),\
   FST::NODE(1, FST::RELATION('o', 7)),\
   FST::NODE(1, FST::RELATION('n', 8)),\
   FST::NODE()

#define GRAPH_CONDITION 4,\
        FST::NODE(1,FST::RELATION('i',1)),\
        FST::NODE(1,FST::RELATION('s',2)),\
        FST::NODE(1,FST::RELATION(':',3)),\
        FST::NODE()

#define GRAPH_NUMBER 7,\
        FST::NODE(1, FST::RELATION('n',1)),\
        FST::NODE(1, FST::RELATION('u',2)),\
        FST::NODE(1, FST::RELATION('m',3)),\
        FST::NODE(1, FST::RELATION('b',4)),\
        FST::NODE(1, FST::RELATION('e',5)),\
```

```
        FST::NODE(1, FST::RELATION('r',6)),\
        FST::NODE()

#define GRAPH_STRING 5, \
        FST::NODE(1, FST::RELATION('l',1)),\
        FST::NODE(1, FST::RELATION('i',2)),\
        FST::NODE(1, FST::RELATION('n',3)),\
        FST::NODE(1, FST::RELATION('e',4)),\
        FST::NODE()

#define GRAPH_CHAR 5, \
        FST::NODE(1, FST::RELATION('c',1)),\
        FST::NODE(1, FST::RELATION('h',2)),\
        FST::NODE(1, FST::RELATION('a',3)),\
        FST::NODE(1, FST::RELATION('r',4)),\
        FST::NODE()

#define GRAPH_MAIN 5, \
        FST::NODE(1,FST::RELATION('m',1)),\
        FST::NODE(1,FST::RELATION('a',2)),\
        FST::NODE(1,FST::RELATION('i',3)),\
        FST::NODE(1,FST::RELATION('n',4)),\
        FST::NODE()

#define GRAPH_CYCLE 6, \
        FST::NODE(1,FST::RELATION('c',1)),\
        FST::NODE(1,FST::RELATION('y',2)),\
        FST::NODE(1,FST::RELATION('c',3)),\
        FST::NODE(1,FST::RELATION('l',4)),\
        FST::NODE(1,FST::RELATION('e',5)),\
        FST::NODE()

#define GRAPH_WRITE 6, \
        FST::NODE(1, FST::RELATION('p',1)),\
        FST::NODE(1, FST::RELATION('r',2)),\
        FST::NODE(1, FST::RELATION('i',3)),\
        FST::NODE(1, FST::RELATION('n',4)),\
        FST::NODE(1, FST::RELATION('t',5)),\
        FST::NODE()

#define GRAPH_NEWLINE 10, \
        FST::NODE(1, FST::RELATION('w',1)),\
        FST::NODE(1, FST::RELATION('r',2)),\
        FST::NODE(1, FST::RELATION('i',3)),\
        FST::NODE(1, FST::RELATION('t',4)),\
```

```
        FST::NODE(1, FST::RELATION('e',5)),\
        FST::NODE(1, FST::RELATION('l',6)),\
        FST::NODE(1, FST::RELATION('i',7)),\
        FST::NODE(1, FST::RELATION('n',8)),\
        FST::NODE(1, FST::RELATION('e',9)),\
        FST::NODE()

#define GRAPH_RETURN 7, \
        FST::NODE(1, FST::RELATION('r',1)),\
        FST::NODE(1, FST::RELATION('e',2)),\
        FST::NODE(1, FST::RELATION('t',3)),\
        FST::NODE(1, FST::RELATION('u',4)),\
        FST::NODE(1, FST::RELATION('r',5)),\
        FST::NODE(1, FST::RELATION('n',6)),\
        FST::NODE()

#define GRAPH_VOID 5, \
        FST::NODE(1, FST::RELATION('p',1)),\
        FST::NODE(1, FST::RELATION('r',2)),\
        FST::NODE(1, FST::RELATION('o',3)),\
        FST::NODE(1, FST::RELATION('c',4)),\
        FST::NODE()

#define GRAPH_VAR 4, \
        FST::NODE(1,FST::RELATION('v',1)),\
        FST::NODE(1,FST::RELATION('a',2)),\
        FST::NODE(1,FST::RELATION('r',3)),\
        FST::NODE()

#define GRAPH_HEX_LITERAL 4, \
   FST::NODE(2, FST::RELATION('-', 0), FST::RELATION('0', 1)), \
   FST::NODE(1, FST::RELATION('x', 2)), \
   FST::NODE(44, \
                FST::RELATION('0', 2), FST::RELATION('1', 2), FST::RELATION('2',
2), FST::RELATION('3', 2), \
       FST::RELATION('4', 2), FST::RELATION('5', 2), FST::RELATION('6', 2),
FST::RELATION('7', 2), \
       FST::RELATION('8', 2), FST::RELATION('9', 2), FST::RELATION('A', 2),
FST::RELATION('B', 2), \
       FST::RELATION('C', 2), FST::RELATION('D', 2), FST::RELATION('E', 2),
FST::RELATION('F', 2), \
       FST::RELATION('a', 2), FST::RELATION('b', 2), FST::RELATION('c', 2),
FST::RELATION('d', 2), \
       FST::RELATION('e', 2), FST::RELATION('f', 2) ,\
```

```
        FST::RELATION('0', 3), FST::RELATION('1', 3), FST::RELATION('2', 3),
FST::RELATION('3', 3), \
        FST::RELATION('4', 3), FST::RELATION('5', 3), FST::RELATION('6', 3),
FST::RELATION('7', 3), \
        FST::RELATION('8', 3), FST::RELATION('9', 3), FST::RELATION('A', 3),
FST::RELATION('B', 3), \
        FST::RELATION('C', 3), FST::RELATION('D', 3), FST::RELATION('E', 3),
FST::RELATION('F', 3), \
        FST::RELATION('a', 3), FST::RELATION('b', 3), FST::RELATION('c', 3),
FST::RELATION('d', 3), \
        FST::RELATION('e', 3), FST::RELATION('f', 3) \
    ), \
    FST::NODE()
```

Листинг 2 — Конечные автоматы для ключевых слов языка

Приложение В

```
------------------------- ТАБЛИЦА ИДЕНТИФИКАТОРОВ -----------------------

|  N  |СТРОКА В ТЛ| ТИП ИДЕНТИФИКАТОРА |        ИМЯ        | ЗНАЧЕНИЕ (ПАРАМЕТРЫ)
|   0 |       2   | number  function   |                          min | P0:number | P1:number |
|   1 |       5   | number parameter   |                         minx |
|   2 |       8   | number parameter   |                         miny |
|   3 |      13   | number  variable   |                       minres |0
|   4 |      41   |   proc  function   |                        stand | P0:line | P1:line |
|   5 |      44   |   line parameter   |                       standa |
|   6 |      47   |   line parameter   |                       standb |
|   7 |      52   |   line  variable   |                     standstr |[0]
|   8 |      56   |   line  LIB FUNC   |                       concat | P0:line | P1:line |
|   9 |      64   |   line   literal   |                        LTRL1 |[8]Result:
|  10 |      77   |   line   literal   |                        LTRL2 |[26]Number at sixteen format:
|  11 |      81   | number  variable   |                      mainsix |74
|  12 |      83   | number   literal   |                        LTRL3 |74
|  13 |      91   |   line   literal   |                        LTRL4 |[22]----------------------
|  14 |      97   |   line  variable   |                     mainstr |[0]
|  15 |     101   | number  variable   |                     mainabc |0
|  16 |     103   | number   literal   |                        LTRL5 |2
|  17 |     107   | number  variable   |                    mainabcd |0
|  18 |     109   | number   literal   |                        LTRL6 |4
|  19 |     113   | number  variable   |                       mainf |0
|  20 |     115   | number   literal   |                        LTRL7 |-4
|  21 |     119   | number  variable   |                       mains |0
|  22 |     125   | number  variable   |                  mainfinish |0
|  23 |     134   |   line   literal   |                        LTRL8 |[20]result of division:
|  24 |     147   |   line   literal   |                        LTRL9 |[1](
|  25 |     153   |   line   literal   |                       LTRL10 |[2]>=
|  26 |     159   |   line   literal   |                       LTRL11 |[3]) :
|  27 |     170   |   line   literal   |                       LTRL12 |[4]True
|  28 |     177   |   line   literal   |                       LTRL13 |[5]False
|  29 |     192   |   line   literal   |                       LTRL14 |[16]Copited string:
|  30 |     196   |   line  variable   |                  standstroka |[0]
|  31 |     198   |   line   literal   |                       LTRL15 |[5]Denis
|  32 |     207   |   line  variable   |                standstrochka |[0]
|  33 |     211   |   line  LIB FUNC   |                      copystr | P0:line |
|  34 |     223   |   char  variable   |                     standmh |[0]
|  35 |     225   |   char   literal   |                       LTRL16 |[1]H
|  36 |     229   |   char  variable   |                     standmi |[0]
|  37 |     231   |   char   literal   |                       LTRL17 |[1]i
|  38 |     235   |   char  variable   |                    standmiv |[0]
|  39 |     237   |   char   literal   |                       LTRL18 |[1]!
|  40 |     257   |   line  variable   |                    standdsa |[0]
|  41 |     263   |   line  variable   |                    standytr |[0]
|  42 |     268   |   line   literal   |                       LTRL19 |[9]Compare:
|  43 |     272   | number  variable   |                    standasd |0
|  44 |     276   | number  LIB FUNC   |                     compare | P0:line | P1:line |
|  45 |     286   | number   literal   |                       LTRL20 |1
|  46 |     291   |   line   literal   |                       LTRL21 |[22]The lines are the same
|  47 |     297   |   line   literal   |                       LTRL22 |[26]The lines are not the same
|  48 |     309   |   line   literal   |                       LTRL23 |[28]Exponentiation of a number:
|  49 |     313   | number  variable   |                       standp |0
|  50 |     315   | number   literal   |                       LTRL24 |6
|  51 |     319   | number  LIB FUNC   |                         poww | P0:number | P1:number |
|  52 |     337   |   line   literal   |                       LTRL25 |[15]Random number:
|  53 |     341   | number  variable   |                       standx |0
|  54 |     345   | number  LIB FUNC   |                          rnd | P0:number | P1:number |
|  55 |     349   | number   literal   |                       LTRL26 |10
|  56 |     364   | number  variable   |                       standu |0
|  57 |     366   | number   literal   |                       LTRL27 |5
|  58 |     370   | number  variable   |                       standv |0
|  59 |     372   | number   literal   |                       LTRL28 |7
|  60 |     376   | number  variable   |                       standr |0
|  61 |     379   |   line   literal   |                       LTRL29 |[24]Arithmetic expression:
|  62 |     398   | number   literal   |                       LTRL30 |8
|  63 |     414   |   line  variable   |                       standc |[0]
|  64 |     416   |   line   literal   |                       LTRL31 |[3]123
|  65 |     420   | number  variable   |                       stande |0
```

Рисунок В.1 – Начало таблицы идентификаторов

```
|  66 |     423   |   line   literal   |                       LTRL32 |[33]Converting a string to a number:
|  67 |     427   | number  LIB FUNC   |                        atoii | P0:line |
|  68 |     444   | number  variable   |                       standk |0
|  69 |     448   |   line  variable   |                     standlen |[0]
|  70 |     450   |   line   literal   |                       LTRL33 |[9]I love SE
|  71 |     454   | number  LIB FUNC   |                      slength | P0:line |
|  72 |     460   |   line   literal   |                       LTRL34 |[13]Line length:
|  73 |     474   | number  variable   |                    standnumb |0
|  74 |     476   | number   literal   |                       LTRL35 |54
|  75 |     480   | number  variable   |               standremainder |0
|  76 |     485   | number   literal   |                       LTRL36 |3
|  77 |     489   |   line   literal   |                       LTRL37 |[34]Remainder after division 54 by 3:
|  78 |     503   | number  variable   |                  standresult |0
|  79 |     507   | number  variable   |                     standsdv |0
|  80 |     518   |   line   literal   |                       LTRL38 |[35]Number after shifted to the left :
|  81 |     532   |   line  variable   |                    standname |[0]
|  82 |     534   |   line   literal   |                       LTRL39 |[6]Denis
|  83 |     538   |   line  variable   |                standsurname |[0]
|  84 |     540   |   line   literal   |                       LTRL40 |[7]Mamonko
|  85 |     544   | number  variable   |                    standpoi |0
|  86 |     550   | number  variable   |                    standisi |0
|  87 |     572   |   line   literal   |                       LTRL41 |[5]Error
|  88 |     583   | number  variable   |                     standab |0
|  89 |     588   |   line   literal   |                       LTRL42 |[20]Cycle from 2 to 10:
|  90 |     601   |   line   literal   |                       LTRL43 |[1]

-------------------------------------------------------------------------
```

Рисунок В.2 – Конец таблицы идентификаторов

---------- ТАБЛИЦА ЛЕКСЕМ ----------

N	ЛЕКСЕМА	СТРОКА	ИНДЕКС В ТИ
0	t	1	
1	f	1	
2	i	1	0
3	(	1	
4	t	1	
5	i	1	1
6	,	1	
7	t	1	
8	i	1	2
9	)	1	
10	[	2	
11	n	3	
12	t	3	
13	i	3	3
14	;	3	
15	?	4	
16	i	4	1
17	>	4	
18	i	4	2
19	#	4	
20	w	5	
21	[	5	
22	i	5	3
23	=	5	
24	i	5	1
25	;	5	
26	]	5	
27	r	6	
28	[	6	
29	i	6	3
30	=	6	
31	i	6	2
32	;	6	
33	]	6	
34	#	6	
35	e	7	
36	i	7	3
37	;	7	
38	]	8	
39	p	10	
40	f	10	
41	i	10	4
42	(	10	
43	t	10	
44	i	10	5
45	,	10	
46	t	10	
47	i	10	6
48	)	10	
49	[	11	
50	n	12	
51	t	12	
52	i	12	7
53	;	12	
54	i	13	7
55	=	13	
56	i	13	8
57	(	13	
58	i	13	5
59	,	13	
60	i	13	6
61	)	13	
62	;	13	
63	o	14	
64	l	14	9
65	;	14	
66	o	15	

Рисунок В.3 – Начало таблицы лексем

547	;	162	
548	n	163	
549	t	163	
550	i	163	86
551	=	163	
552	l	163	18
553	;	163	
554	?	164	
555	i	164	85
556	@	164	
557	i	164	86
558	#	164	
559	w	165	
560	[	165	
561	i	165	4
562	(	165	
563	i	165	81
564	,	165	
565	i	165	83
566	)	165	
567	;	165	
568	]	165	
569	r	166	
570	[	166	
571	o	166	
572	l	166	87
573	;	166	
574	]	166	
575	#	166	
576	o	167	
577	l	167	13
578	;	167	
579	^	168	
580	;	168	
581	n	171	
582	t	171	
583	i	171	88
584	=	171	
585	l	171	16
586	;	171	
587	o	172	
588	l	172	89
589	;	172	
590	?	173	
591	i	173	88
592	!	173	
593	l	173	55
594	#	173	
595	c	174	
596	[	174	
597	o	175	
598	i	175	88
599	;	175	
600	o	176	
601	l	176	90
602	;	176	
603	i	177	88
604	=	177	
605	i	177	88
606	l	177	16
607	+	177	
608	;	177	
609	]	178	
610	#	178	
611	o	179	
612	i	179	88
613	;	179	
614	^	180	
615	;	180	
616	]	181	

Рисунок В.4 – Конец таблицы лексем

Приложение Г

Таблица 4.1 – Таблица правил переходов нетерминальных символов

Символ	Правила	Описание
S	S->tfiPTS S->pfiPGS S->m[K]	Стартовые правила, описывающие общую структуру программы
P	P->(E) P->()	Правила списка параметров функции
E	E->ti,E E->ti	Правила для параметров функции при её объявлении
T	T->[KeV;] T->[eV;]	Правила для тела функции
K	K->nti;K K->?Z#R K->i=W;K K->oV;K K->^; K->nti=V;K K->^;K K->?Z#RK K->iF;K K->nti=V; K->i=W; K->nti; K->oV; K->iF;	Правила для конструкций внутри функций
G	G->[e;] G->[Ke;]	Правила для процедуры
F	F->(N) F->()	Правила для списка параметров, передаваемых в функцию
N	N->i N->l N->l,N N->i,N	Правила для параметров, передаваемых в функцию
R	R->rY# R->wY# R->cY# R->rYwY# R->wYrY#	Правила для условных конструкций и цикла
Y	Y->[X]	Правило, описывающее тело функции/условного выражения
Z	Z->iLi Z->iLl Z->lLi	Правила, описывающее условие цикла/условного выражения

Таблица 4.1 (Продолжение)

L	L-> L->> L->& L->! L->~ L->@	Правила, описывающие условные операторы
A	A->+ A->- A->* A->/ A->% A->} A->{	Правила, описывающие условные операторы и операторы сдвигов
V	V->l V->i V->h	Правила для выражений
W	W->i W->l W->(W) W->(W)AW W->iF W->iAW W->Law W->iFAW	Правила, описывающие арифметические выражения
X	X->i=W;X X->0V;X X->^;X X->iF;X X->i=W; X->oV; X->^; X->iF;	Правила, описывающие синтаксические конструкции цикла/условного выражения

Приложение Д

```
namespace GRB
{
   struct Rule
   {
      GRBALPHABET  nn;
      int iderror;
      short size;

      struct Chain
      {
         short size;
         GRBALPHABET* nt;
         Chain() { size = 0; nt = 0; };
         Chain(
            short psize,
            GRBALPHABET s, ...
         );
         char* getCChain(char* b);
         static GRBALPHABET T(char t) { return GRBALPHABET(t); };
         static GRBALPHABET N(char n) { return -GRBALPHABET(n); };
         static bool isT(GRBALPHABET s) { return s > 0; };
         static bool isN(GRBALPHABET s) { return !isT(s); };
         static char alphabet_to_char(GRBALPHABET s) { return isT(s) ? char(s) :
char(-s); };
      }*chains;

      Rule() { nn = 0x00; size = 0; }
      Rule(
         GRBALPHABET pnn,
         int iderror,
         short psize,
         Chain c, ...
      );
      char* getCRule(
         char* b,
         short nchain
      );
      short getNextChain(
         GRBALPHABET t,
         Rule::Chain& pchain,
         short j
      );
   };
```

```
    struct Greibach
    {
        short size;
        GRBALPHABET startN;
        GRBALPHABET stbottomT;
        Rule* rules;
        Greibach() { short size = 0; startN = 0; stbottomT = 0; rules = 0; };
        Greibach(
            GRBALPHABET pstartN,
            GRBALPHABET pstbootomT,
            short psize,
            Rule r, ...
        );
        short getRule(
            GRBALPHABET pnn,
            Rule& prule
        );
        Rule getRule(short n);
    };
    Greibach getGreibach();
};
```

Листинг 3 – Структура грамматики Грейбах

```
namespace MFST
{
    struct MfstState
    {
        short lenta_position;
        short nrule;
        short nrulechain;
        MFSTSTSTACK st;
        MfstState();
        MfstState(
            short pposition,
            MFSTSTSTACK pst,
            short pnrulechain
        );
        MfstState(
            short pposition,
            MFSTSTSTACK pst,
            short pnrule,
            short pnrulechain
        );
    };
```

```
struct Mfst
{
   enum RC_STEP
   {
      NS_OK,
      NS_NORULE,
      NS_NORULECHAIN,
      NS_ERROR,
      TS_OK,
      TS_NOK,
      LENTA_END,
      SURPRISE
   };
   struct MfstDiagnosis
   {
      short lenta_position;
      RC_STEP rc_step;
      short nrule;
      short nrule_chain;
      MfstDiagnosis();
      MfstDiagnosis(
         short plenta_position,
         RC_STEP prc_step,
         short pnrule,
         short pnrule_chain
      );
   }
   diagnosis[MFST_DIAGN_digit];

   class my_stack_MfstState :public std::stack<MfstState> {
   public:
      using std::stack<MfstState>::c;
   };

   GRBALPHABET* lenta;
   short lenta_position;
   short nrule;
   short nrulechain;
   short lenta_size;
   GRB::Greibach grebach;
   Lexer::LEX lex;
   MFSTSTSTACK st;
   my_stack_MfstState storestate;
   Mfst();
```

```
        Mfst(
            Lexer::LEX plex,
            GRB::Greibach pgrebach
        );
        char* getCSt(char* buf);
        char* getCLenta(char* buf, short pos, short n = 25);
        char* getDiagnosis(short n, char* buf);
        bool savestate(const Log::LOG& log);
        bool reststate(const Log::LOG& log);
        bool push_chain(
            GRB::Rule::Chain chain
        );
        RC_STEP step(const Log::LOG& log);
        bool start(const Log::LOG& log);
        bool savediagnois(
            RC_STEP pprc_step
        );
        void printrules(const Log::LOG& log);
        struct Deducation
        {
            short size;
            short* nrules;
            short* nrulechains;
            Deducation() { size = 0; nrules = 0; nrulechains = 0; };
        } deducation;
        bool savededucation();
    };
};
```

Листинг 4 – Структура магазинного конечного автомата

Приложение Е

```
0   : S->tfiPTS
3   : P->(E)
4   : E->ti,E
7   : E->ti
10  : T->[KeV;]
11  : K->nti;K
15  : K->?Z#R
16  : Z->iLi
17  : L->>
20  : R->wYrY#
21  : Y->[X]
22  : X->i=W;
24  : W->i
28  : Y->[X]
29  : X->i=W;
31  : W->i
36  : V->i
39  : S->pfiPGS
42  : P->(E)
43  : E->ti,E
46  : E->ti
49  : G->[Ke;]
50  : K->nti;K
54  : K->i=W;K
56  : W->iF
57  : F->(N)
58  : N->i,N
60  : N->i
63  : K->oV;K
64  : V->l
66  : K->oV;K
67  : V->i
69  : K->^;
74  : S->m[K]
76  : K->oV;K
77  : V->l
79  : K->nti=V;K
83  : V->l
85  : K->oV;K
86  : V->i
88  : K->^;K
90  : K->oV;K
91  : V->l
93  : K->^;K
95  : K->nti;K
99  : K->nti=V;K
103 : V->l
105 : K->nti=V;K
109 : V->l
111 : K->nti=V;K
115 : V->l
117 : K->nti=V;K
121 : V->l
123 : K->nti;K
127 : K->i=W;K
129 : W->iAW
130 : A->/
```

Рисунок Е.1 – Начало дерева разбора

```
592 : K->?Z#RK
593 : Z->iLl
594 : L->!
597 : R->cY#
598 : Y->[X]
599 : X->oV;X
600 : V->i
602 : X->oV;X
603 : V->l
605 : X->i=W;
607 : W->iAW
608 : A->+
609 : W->l
613 : K->oV;K
614 : V->i
616 : K->^;
```

Рисунок Е.2 – Конец дерева разбора

Приложение Ж

```
----------------  ЛЕКСЕМЫ СООТВЕТСТВУЮЩИЕ ИСХОДНОМУ КОДУ ---------------------

 1 | tfi[0](ti[1],ti[2])
 2 | [
 3 | nti[3];
 4 | ?i[1]>i[2]#
 5 | w[i[3]=i[1];]
 6 | r[i[3]=i[2];]#
 7 | ei[3];
 8 | ]
10 | pfi[4](ti[5],ti[6])
11 | [
12 | nti[7];
13 | i[7]=i[8](i[5],i[6]);
14 | ol[9];
15 | oi[7];
16 | ^;
17 | e;
18 | ]
20 | m[
21 | ol[10];
22 | nti[11]=l[12];
23 | oi[11];
24 | ^;
25 | ol[13];
26 | ^;
29 | nti[14];
30 | nti[15]=l[16];
31 | nti[17]=l[18];
34 | nti[19]=l[20];
35 | nti[21]=l[16];
36 | nti[22];
37 | i[22]=i[19]i[21]/;
38 | ol[23];
39 | oi[22];
40 | ^;
41 | ol[13];
42 | ^;
45 | ol[24];oi[15];ol[25];oi[17];ol[26];
46 | ?i[15]~i[17]#
47 | w[i[7]=l[27];]
48 | r[i[7]=l[28];]#
49 | oi[7];
50 | ^;
51 | ol[13];
52 | ^;
55 | ol[29];
56 | nti[30]=l[31];
57 | oi[30];
58 | ^;
59 | nti[32];
60 | i[32]=i[33](i[30]);
61 | ol[13];
62 | ^;
65 | nti[34]=l[35];
66 | nti[36]=l[37];
67 | nti[38]=l[39];
68 | oi[34];
69 | oi[36];
70 | oi[38];
71 | ^;
72 | ol[13];
73 | ^;
76 | nti[40]=l[28];
77 | nti[41]=l[28];
78 | ol[42];
79 | nti[43];
80 | i[43]=i[44](i[40],i[41]);
81 | ?i[43]&l[45]#
82 | w[ol[46];]
83 | r[ol[47];]#
```

Рисунок Ж.1 – Промежуточное представление кода (начало)

```
 94 | ol[13];
 95 | ^;
 98 | ol[52];
 99 | nti[53];
100 | i[53]=i[54](l[45],l[55]);
101 | oi[53];
102 | ^;
103 | ol[13];
104 | ^;
107 | nti[56]=l[57];
108 | nti[58]=l[59];
109 | nti[60];
110 | ol[61];
111 | i[60]=i[56]i[58]l[16]*+l[18]l[18]+l[16]/-i[0](l[57],l[62])+;
112 | oi[60];
113 | ^;
114 | ol[13];
115 | ^;
118 | nti[63]=l[64];
119 | nti[65];
120 | ol[66];
121 | i[65]=i[67](i[63]);
122 | oi[65];
123 | ^;
124 | ol[13];
125 | ^;
128 | nti[68];
129 | nti[69]=l[70];
130 | i[68]=i[71](i[69]);
131 | ol[72];
132 | oi[68];
133 | ^;
134 | ol[13];
135 | ^;
138 | nti[73]=l[74];
139 | nti[75];
140 | i[75]=i[73]l[76]%;
141 | ol[77];
142 | oi[75];
143 | ^;
144 | ol[13];
145 | ^;
148 | nti[78];
149 | nti[79]=l[76];
150 | i[78]=i[79]l[16]{;
151 | ol[80];
152 | oi[78];
153 | ^;
154 | ol[13];
155 | ^;
158 | nti[81]=l[82];
159 | nti[83]=l[84];
162 | nti[85]=l[18];
163 | nti[86]=l[18];
164 | ?i[85]@i[86]#
165 | w[i[4](i[81],i[83]);]
166 | r[ol[87];]#
167 | ol[13];
168 | ^;
171 | nti[88]=l[16];
172 | ol[89];
173 | ?i[88]!l[55]#
174 | c[
175 | oi[88];
176 | ol[90];
177 | i[88]=i[88]l[16]+;
178 | ]#
179 | oi[88];
180 | ^;
181 | ]
```

Рисунок Ж.2 – Промежуточное представление кода (конец)

Приложение 3

```
.586
.model flat, stdcall
includelib libucrt.lib
includelib kernel32.lib
includelib "../Debug/StaticLibrary.lib
ExitProcess PROTO:DWORD
.stack 4096


 outlich PROTO : DWORD

 outrad PROTO : DWORD

 concat PROTO : DWORD, : DWORD, : DWORD

 poww PROTO : DWORD, : DWORD, : DWORD

 compare PROTO : DWORD, : DWORD, : DWORD

 rnd PROTO : DWORD, : DWORD, : DWORD

 slength PROTO : DWORD, : DWORD

 atoii  PROTO : DWORD,  : DWORD

 copystr PROTO : DWORD, : DWORD

.const
                newline byte 13, 10, 0
                LTRL1 byte 'Result: ', 0
                LTRL2 byte 'Number at sixteen format: ', 0
                LTRL3 sdword 74
                LTRL4 byte '----------------------', 0
                LTRL5 sdword 2
                LTRL6 sdword 4
                LTRL7 sdword -4
                LTRL8 byte 'result of division: ', 0
                LTRL9 byte '(', 0
                LTRL10 byte '>=', 0
                LTRL11 byte ') :', 0
                LTRL12 byte 'True', 0
                LTRL13 byte 'False', 0
                LTRL14 byte 'Copited string: ', 0
```

```
        LTRL15 byte 'Denis', 0
        LTRL16 byte 'H', 0
        LTRL17 byte 'i', 0
        LTRL18 byte '!', 0
        LTRL19 byte 'Compare: ', 0
        LTRL20 sdword 1
        LTRL21 byte 'The lines are the same', 0
        LTRL22 byte 'The lines are not the same', 0
        LTRL23 byte 'Exponentiation of a number: ', 0
        LTRL24 sdword 6
        LTRL25 byte 'Random number: ', 0
        LTRL26 sdword 10
        LTRL27 sdword 5
        LTRL28 sdword 7
        LTRL29 byte 'Arithmetic expression:  ', 0
        LTRL30 sdword 8
        LTRL31 byte '123', 0
        LTRL32 byte 'Converting a string to a number: ', 0
        LTRL33 byte 'I love SE', 0
        LTRL34 byte 'Line length: ', 0
        LTRL35 sdword 54
        LTRL36 sdword 3
        LTRL37 byte 'Remainder after division 54 by 3: ', 0
        LTRL38 byte 'Number after shifted to the left : ', 0
        LTRL39 byte 'Denis ', 0
        LTRL40 byte 'Mamonko', 0
        LTRL41 byte 'Error', 0
        LTRL42 byte 'Cycle from 2 to 10: ', 0
        LTRL43 byte ' ', 0
.data
        temp sdword ?
        buffer byte 256 dup(0)
        minres dword 0
        standstr dword ?
        mainsix dword 0
        mainstr dword ?
        mainabc dword 0
        mainabcd dword 0
        mainf dword 0
        mains dword 0
        mainfinish dword 0
        standstroka dword ?
        standstrochka dword ?
        standmh dword ?
        standmi dword ?
```

```
                standmiv dword ?
                standdsa dword ?
                standytr dword ?
                standasd dword 0
                standp dword 0
                standx dword 0
                standu dword 0
                standv dword 0
                standr dword 0
                standc dword ?
                stande dword 0
                standk dword 0
                standlen dword ?
                standnumb dword 0
                standremainder dword 0
                standresult dword 0
                standsdv dword 0
                standname dword ?
                standsurname dword ?
                standpoi dword 0
                standisi dword 0
                standab dword 0
.code

;---------- min ------------
min PROC,
        minx : sdword, miny : sdword
; --- save registers ---
push ebx
push edx
; ---------------------
mov edx, minx
cmp edx, miny

jg right1
jl wrong1
right1:
push minx

pop ebx
mov minres, ebx

jmp next1
wrong1:
push miny
```

```
pop ebx
mov minres, ebx

next1:
; --- restore registers ---
pop edx
pop ebx
; ------------------------
mov eax, minres
ret
min ENDP
;-----------------------------


;---------- stand ------------
stand PROC,
        standa : dword, standb : dword
; --- save registers ---
push ebx
push edx
; ----------------------

push standb
push standa
push offset buffer
call concat
mov standstr, eax

push offset LTRL1
call outrad


push standstr
call outrad

push offset newline
call outrad

; --- restore registers ---
pop edx
pop ebx
; ------------------------
ret
stand ENDP
```

```
;------------------------------


;----------- MAIN ------------
main PROC

push offset LTRL2
call outrad

push LTRL3

pop ebx
mov mainsix, ebx


push mainsix
call outlich

push offset newline
call outrad


push offset LTRL4
call outrad

push offset newline
call outrad

push LTRL5

pop ebx
mov mainabc, ebx

push LTRL6

pop ebx
mov mainabcd, ebx

push LTRL7

pop ebx
mov mainf, ebx

push LTRL5
```

```
pop ebx
mov mains, ebx

push mainf
push mains
pop ebx
pop eax
cdq
idiv ebx
push eax

pop ebx
mov mainfinish, ebx


push offset LTRL8
call outrad


push mainfinish
call outlich

push offset newline
call outrad


push offset LTRL4
call outrad

push offset newline
call outrad


push offset LTRL9
call outrad


push mainabc
call outlich


push offset LTRL10
call outrad
```

```
push mainabcd
call outlich


push offset LTRL11
call outrad

mov edx, mainabc
cmp edx, mainabcd

jz right2
jg right2
jnz wrong2
right2:
mov standstr, offset LTRL12
jmp next2
wrong2:
mov standstr, offset LTRL13
next2:

push standstr
call outrad

push offset newline
call outrad


push offset LTRL4
call outrad

push offset newline
call outrad


push offset LTRL14
call outrad

mov standstroka, offset LTRL15

push standstroka
call outrad

push offset newline
call outrad
```

```
push standstroka
push offset buffer
call copystr
mov standstrochka, eax

push offset LTRL4
call outrad

push offset newline
call outrad

mov standmh, offset LTRL16
mov standmi, offset LTRL17
mov standmiv, offset LTRL18

push standmh
call outrad


push standmi
call outrad


push standmiv
call outrad

push offset newline
call outrad


push offset LTRL4
call outrad

push offset newline
call outrad

mov standdsa, offset LTRL13
mov standytr, offset LTRL13

push offset LTRL19
call outrad


push standytr
```

```
push standdsa
push offset buffer
call compare
push eax

pop ebx
mov standasd, ebx

mov edx, standasd
cmp edx, LTRL20

jz right3
jnz wrong3
right3:

push offset LTRL21
call outrad

jmp next3
wrong3:

push offset LTRL22
call outrad

next3:
push offset newline
call outrad


push offset LTRL4
call outrad

push offset newline
call outrad


push offset LTRL23
call outrad

push LTRL24

pop ebx
mov standp, ebx
```

```
push standp
push LTRL5
push offset buffer
call poww
push eax

pop ebx
mov standp, ebx


push standp
call outlich

push offset newline
call outrad


push offset LTRL4
call outrad

push offset newline
call outrad


push offset LTRL25
call outrad


push LTRL26
push LTRL20
push offset buffer
call rnd
push eax

pop ebx
mov standx, ebx


push standx
call outlich

push offset newline
call outrad
```

```
push offset LTRL4
call outrad

push offset newline
call outrad

push LTRL27

pop ebx
mov standu, ebx

push LTRL28

pop ebx
mov standv, ebx


push offset LTRL29
call outrad

push standu
push standv
push LTRL5
pop ebx
pop eax
imul eax, ebx
push eax
pop ebx
pop eax
add eax, ebx
push eax
push LTRL6
push LTRL6
pop ebx
pop eax
add eax, ebx
push eax
push LTRL5
pop ebx
pop eax
cdq
idiv ebx
push eax
pop ebx
pop eax
```

```
sub eax, ebx
jnc bk
neg eax
bk:
push eax

push LTRL30
push LTRL27
call min
push eax
pop ebx
pop eax
add eax, ebx
push eax

pop ebx
mov standr, ebx


push standr
call outlich

push offset newline
call outrad


push offset LTRL4
call outrad

push offset newline
call outrad

mov standc, offset LTRL31

push offset LTRL32
call outrad


push standc
push offset buffer
call atoii
push eax

pop ebx
mov stande, ebx
```

```
push stande
call outlich

push offset newline
call outrad


push offset LTRL4
call outrad

push offset newline
call outrad

mov standlen, offset LTRL33

push standlen
push offset buffer
call slength
push eax

pop ebx
mov standk, ebx


push offset LTRL34
call outrad


push standk
call outlich

push offset newline
call outrad


push offset LTRL4
call outrad

push offset newline
call outrad

push LTRL35
```

```
pop ebx
mov standnumb, ebx

push standnumb
push LTRL36
pop ebx
pop eax
cdq
mov edx,0
idiv ebx
push edx

pop ebx
mov standremainder, ebx


push offset LTRL37
call outrad


push standremainder
call outlich

push offset newline
call outrad


push offset LTRL4
call outrad

push offset newline
call outrad

push LTRL36

pop ebx
mov standsdv, ebx

push standsdv
push LTRL5
pop ebx
pop eax
mov cl, bl
shl eax, cl
push eax
```

```
pop ebx
mov standresult, ebx


push offset LTRL38
call outrad


push standresult
call outlich

push offset newline
call outrad


push offset LTRL4
call outrad

push offset newline
call outrad

mov standname, offset LTRL39
mov standsurname, offset LTRL40
push LTRL6

pop ebx
mov standpoi, ebx

push LTRL6

pop ebx
mov standisi, ebx

mov edx, standpoi
cmp edx, standisi

jz right4
jl right4
jnz wrong4
right4:

push standsurname
push standname
call stand
```

```
jmp next4
wrong4:

push offset LTRL41
call outrad

next4:

push offset LTRL4
call outrad

push offset newline
call outrad

push LTRL5

pop ebx
mov standab, ebx


push offset LTRL42
call outrad

mov edx, standab
cmp edx, LTRL26

jnz cycle5
jmp cyclenext5
cycle5:

push standab
call outlich


push offset LTRL43
call outrad

push standab
push LTRL5
pop ebx
pop eax
add eax, ebx
push eax
```

```
pop ebx
mov standab, ebx

mov edx, standab
cmp edx, LTRL26

jnz cycle5
cyclenext5:

push standab
call outlich

push offset newline
call outrad

push 0
call ExitProcess
main ENDP
end main
```

Листинг 5 – Результат генерации ассемблерного кода